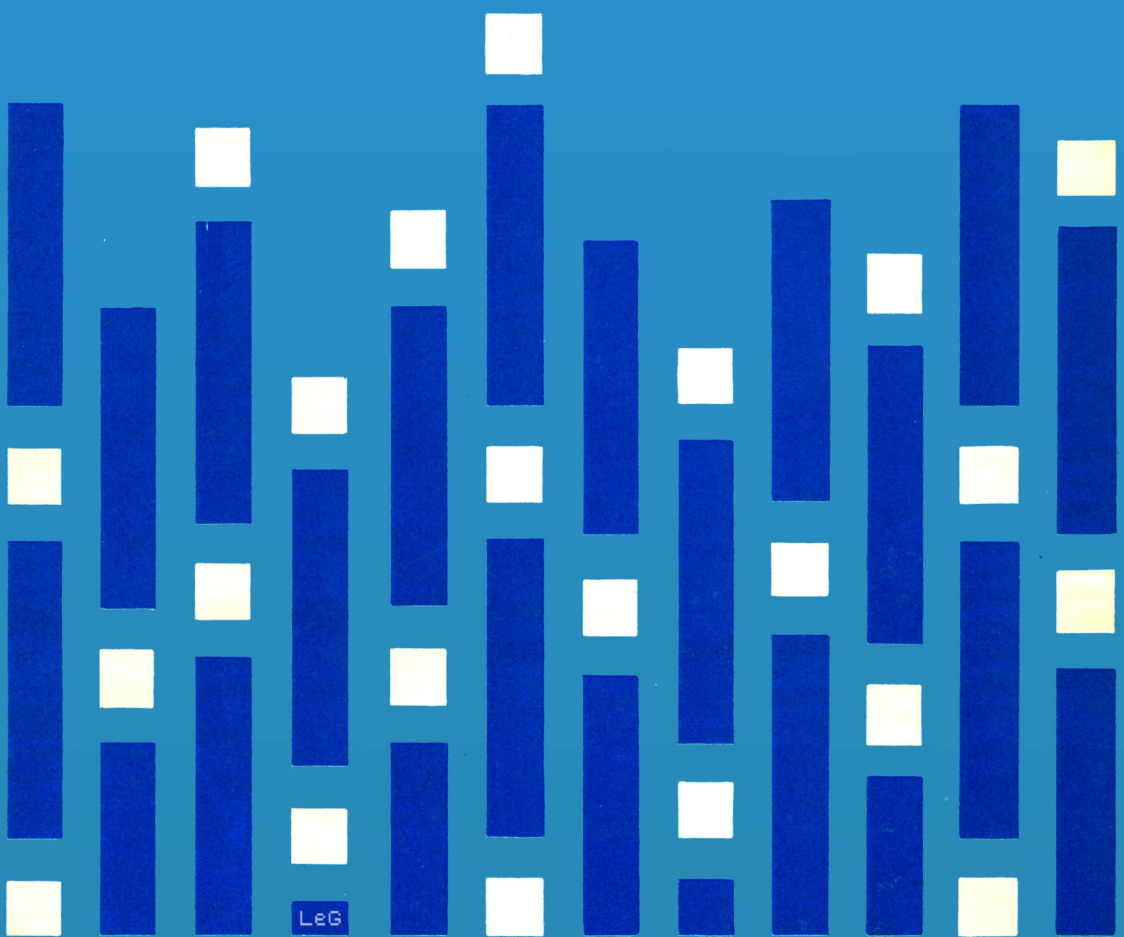
CURSO DE INFORMATICA

INTRODUCCION AL BASIC

A. POZO



Alhambra

CURSO DE INFORMATICA

Introducción al BASIC

EDITORIAL ALHAMBRA

CURSO DE INFORMATICA

Precio: 1.200 ptas.

EDITORIAL ALHAMBRA

CURSO DE INFORMATICA

Precio: 1.200 ptas.

A. POZO CHIA
Catedrático de Matemáticas

CURSO DE INFORMATICA

Introducción al BASIC

Alhambra

Primera edición, 1983

© EDITORIAL ALHAMBRA, S. A.
R. E. 182
Madrid-1. Claudio Coello, 76

Delegaciones:

Barcelona-8. Enrique Granados, 61
Bilbao-14. Doctor Albiñana, 12
Granada. Pza. de las Descalzas, 2
La Coruña-5. Pasadizo de Pernas, 13
Madrid-2. Saturnino Calleja, 1
Oviedo-6. Avda. del Cristo, 9
Santa Cruz de Tenerife. General Porlier, 14
Sevilla-12. Reina Mercedes, 35
Valencia-3. Cabillers, 5
Zaragoza-5. Concepción Arenal, 25

México

Editorial Alhambra Mexicana, S. A.
Avda. División del Norte, 2412
03340 México, D. F.

n c 06090290

© Es propiedad del autor

Reservados todos los derechos. Ni la totalidad, ni parte de este libro pueden reproducirse o transmitirse, utilizando medios electrónicos o mecánicos, por fotocopia, grabación, información, anulado, u otro sistema, sin permiso por escrito del editor.

Cubierta: Antonio Tello

ISBN 84-205-0986-8

Depósito legal: M. 21311-1983

Impreso en España - Printed in Spain

Selecciones Gráficas - Carretera de Irún, km. 11,500 - Madrid (1983)

Indice

Capítulo *páginas*

Prólogo VII

PRIMERA PARTE
INTRODUCCION A LOS ORDENADORES

1 Desarrollo histórico 3

1.1. Introducción, 3. 1.2. Los orígenes, 3. 1.3. Primeros soportes de información, 4. 1.4. El ábaco, 5. 1.5. El papel y la imprenta, 6. 1.6. El sistema decimal, 6. 1.7. El aritmómetro de Pascal y el calculador de Leibnitz, 8. 1.8. La ficha perforada, 9. 1.9. La máquina analítica de Babbage, 11. 1.10. La tabuladora de Hollerith, 12. 1.12. Almacenamiento de programas: Von Newman, 13. 1.13. El ENIAC, 13. 1.14. Generaciones de ordenadores, 14. 1.15. Clasificación de los ordenadores, 16.

2 Sistema de numeración binario, octal y hexadecimal 17

2.1. Sistema de numeración base *n*, 17. 2.2. Operaciones con números en base *n*, 21. 2.3. El sistema binario, 25. 2.4. Numeración octal y hexadecimal, 31. 2.5. Ejercicios, 33.

3 El sistema binario en ordenadores 35

3.1. Un diseño muy simple, 35. 3.2. Teoría de circuitos lógicos, 36. 3.3. Algebra de Boole de los conmutadores, 36. 3.4. Ejercicios, 39.

4 Memoria y control de errores. Soportes de información 43

4.1. La memoria, 43. 4.2. El control de errores, 47. 4.3. Soportes de información, 48.

5 Esquema general de un ordenador 54

5.1. Esquema de un ordenador, 54. 5.2. Simulación de un ordenador en clase, 56.

SEGUNDA PARTE
ALGORITMOS Y ORGANIGRAMAS. ESTRUCTURAS BASICAS

6 Algoritmos y organigramas. Lenguajes algorítmicos 58

6.1. Introducción, 58. 6.2. Algoritmos, 59. Ejercicios, 61. 6.3. Organigramas, 62. Ejercicios, 68. 6.4. Lenguajes algorítmicos para ordenadores, 68. Ejercicios, 73.

7 Instrucciones algorítmicas básicas 75

7.1. Introducción, 75. 7.2. Algunos procesos simples sin ruptura de secuencia, 75. 7.3. Saltos incondicionales, 80. 7.4. Decisiones, 83. Ejercicios, 90.

Capítulo *Páginas*

8 Aplicaciones elementales 92

8.1. Introducción, 92. 8.2. Sucesiones, 93. 8.3. Elecciones y ordenaciones, 96. 8.4. Factoriales, 101. 8.5. Divisibilidad, 105.

TERCERA PARTE
COMPLEMENTOS DE BASIC. APLICACIONES

9 Expresiones aritméticas y lógicas 112

9.1. Introducción, 112. Ejercicios, 113. 9.2. Expresiones lógicas, 114. Ejercicios, 116.

10 Instrucciones elementales 118

10.1. Introducción, 118. 10.2. Instrucción INPUT, 120. Ejercicios, 121. 10.3. Instrucción PRINT, 122. Ejercicios, 128. 10.4. Instrucción END, 129. 10.5. Instrucción GO TO, 130. Ejercicios, 133. 10.6. Instrucción IF, 133. Ejercicios, 135. 10.7. Instrucción REM, 136. 10.8. Instrucción LIST, 139. 10.9. Correcciones a un programa, 139. Ejercicios, 140.

11 Ciclos FOR-NEXT 143

11.1. Ciclos simples, 143. 11.2. Ciclos anidados, 149. Ejercicios, 153.

12 Variables subindicadas 156

12.1. Variables con un subíndice, 156. 12.2. Variables con dos o más subíndices, 164. Ejercicios, 167.

13 Variables alfanuméricas 171

13.1. Cadenas, 171. 13.2. Código ASCII, 173. 13.3. Funciones para cadenas, 175. Ejercicios, 180.

CUARTA PARTE
OTRAS INSTRUCCIONES

14 Funciones 184

14.1. Funciones definidas por el sistema operativo, 184. 14.2. Funciones definidas por el usuario, 187. Ejercicios, 188.

15 Subrutinas 190

15.1. Subrutinas, 190. Ejercicios, 194.

16 Las instrucciones READ, DATA y RESTORE 195

16.1. Las instrucciones READ y DATA, 195. 16.2. La instrucción RESTORE, 198. Ejercicios, 199.

17 Ficheros 201

17.1. Generalidades sobre ficheros, 201. 17.2. Tratamiento de ficheros, 202. 17.3. Ficheros de acceso secuencial, 203. 17.4. Ficheros de acceso directo, 204.

Prólogo

Hasta hace muy pocos años, la existencia de ordenadores era conocida por un gran número de personas, si bien, muchas de ellas no habían manejado ninguno, siendo muy pocos los que tenían posibilidades de acceder a algún modelo; normalmente, eran máquinas de grandes dimensiones que encerraban posibilidades misteriosas para cualquiera no versado en la materia.

Los recientes avances de la ciencia física que han permitido sustituir las lámparas de vacío por transistores, y éstos a su vez, por diminutos circuitos integrados, han motivado una reducción de tamaño hasta tal punto que, hoy día, se habla de ordenadores de bolsillo y, en algunos casos, el tamaño es adecuado a la expresión.

Esta reducción de tamaño, acompañada de una reducción de precios, ha contribuido a que su uso sea cada vez más general en todo tipo de actividades: investigación, gestión, lúdicas, etc. Un campo que actualmente está siendo invadido es la enseñanza secundaria, aunque lo más frecuente en este nivel haya sido el uso de monitores para facilitar la didáctica de determinadas materias, es decir, la enseñanza asistida por ordenador.

Además de esta actividad, estimamos necesario ofrecer al estudiante la posibilidad de iniciarse en el mundo de la informática y sus aplicaciones, como unos aspectos importantes de la sociedad en que vivimos.

Un primer objetivo para hacer posible esta iniciación, consistirá en conseguir que el lector adopte una perspectiva histórica sobre la evolución de las máquinas de calcular, que le permita comprender con mayor facilidad la situación actual. Hecha esta introducción histórica pasaremos a estudiar cuáles son los fundamentos de los actuales ordenadores, sus diversos componentes y para qué sirven.

Con respecto a las aplicaciones, debemos pensar que el tratamiento de un problema por ordenador lleva implícito los siguientes puntos:

a) *Conocimiento profundo del problema.*
b) *Construcción de un algoritmo que permita llegar a su solución, es decir, diseñar un camino por el que, paso a paso, nos permita alcanzar los resultados.*
c) *En ciertos casos, para clarificar el apartado* b, *hacer una representación gráfica del algoritmo, que llamaremos organigrama.*
d) *Escribir el algoritmo en un lenguaje que entienda el ordenador.*
e) *Pasarlo por el ordenador y recoger los resultados.*

La parte constituida por los apartados a, b *y* c, *la llamaremos análisis del problema, y será para nosotros la más importante, porque al hacerse independientemente del tipo de ordenador que usemos, tiene un carácter de permanencia en el tiempo, del que pueden carecer los apartados* d *y* e. *Esta idea la tendremos muy presente a lo largo de todo el libro.*

No obstante, el comprobar el funcionamiento del algoritmo construido es siempre una garantía de que no nos hemos equivocado, a la vez que un ejercicio de programación en algún lenguaje de ordenador, por lo que también haremos la traducción de los algoritmos estudiados, a algún lenguaje de uso generalizado.

De los lenguaje algorítmicos para ordenadores, el BASIC, como la propia palabra indica «Beginner's All-purpose Symbolic Instruction Code», esto es, «Código de Instrucciones Simbólicas de Carácter General para Principiantes», es uno de los lenguajes más simples, de alto nivel, presente en la mayoría de los ordenadores y que en enseñanza es usado en un elevado tanto por ciento de los casos por su gran facilidad y versatilidad. Puede enseñarse sin estar limitado a un ordenador concreto, como veremos, adaptándose luego el usuario a cualquier ordenador con suma facilidad.

A pesar de su simpleza, veremos cómo se podrá acceder a la ejecución de algoritmos de un cierto grado de complejidad, así como, con gran rapidez, iniciarse en otros lenguajes de programación, dado que siempre existen determinadas estructuras algorítmicas básicas que se repiten en todos ellos. Será el lenguaje BASIC el que, en sus aspectos más generales, usemos para la programación de los algoritmos planteados.

Finalmente, decir que en cuanto a la designación de las mismas máquinas nos referiremos a ellas por «ordenadores», aunque son de uso frecuente palabras como, computadores, miniordenadores, microordenadores, procesadores, cerebros electrónicos, etc., sin que exista en muchos casos una clara diferencia entre ellas. En el texto, la palabra ordenador, podría ser sustituida por cualquiera de las anteriores, sin que se altere en lo más mínimo su contenido.

PRIMERA PARTE

INTRODUCCION A LOS ORDENADORES

1

Desarrollo histórico

1.1. INTRODUCCION

El desarrollo espectacular de los ordenadores en los últimos años, no debe hacernos pensar que el intento de automatizar los cálculos y de registrar información, por parte del hombre, es relativamente reciente; desde siempre, éste ha estado forzado a un lento y continuo registro de las observaciones y conocimientos que iba adquiriendo y a realizar cálculos tan laboriosos y molestos, que nunca ha desistido en su intento de inventar mecanismos que hicieran más soportable este trabajo.

Los ordenadores son la manifestación más reciente de tales investigaciones, culminando así un proceso iniciado hace miles de años con el uso de piedras o perlas, cajas de arena y ábacos.

Actualmente, los ordenadores combinan poderosamente la rapidez en las operaciones y la capacidad de tratamiento de grandes volúmenes de datos.

Hagamos una breve revisión histórica y sepamos cuáles fueron los predecesores de los modernos ordenadores.

1.2. LOS ORIGENES

Desde el principio de la existencia del hombre, se encontraron procedimientos para dejar constancia de sus realizaciones; la transmisión oral de padres a hijos era suficiente, en la mayoría de los casos, para conservar todos los conocimientos de la época; muy pronto, dada la limitación de la memoria humana, los hombres tuvieron que emplear símbolos sobre algún soporte material, para registrar la información que iban tomando del universo.

En sus cálculos, el hombre primitivo, usaba los dedos de las manos (5 ó 10, manejando sin sospecharlo siquiera, el sistema quinario y decimal) y el seguir después de 10 planteaba serios problemas, resolviéndose en algunos casos pidiendo la colaboración de otras personas.

El siguiente avance fue dado por las civilizaciones antiguas de Egipto (3400 años a. de C., aproximadamente) y otros pueblos de Oriente; consistía en representar los números por piedras («calculus» en latín de donde procede la palabra cálculo), perlas, palillos, etc.

1.3. PRIMEROS SOPORTES DE INFORMACION

Hacia el año 2600 a. de C. los conocimientos acumulados se hacían difíciles de mantener en la memoria y mediante símbolos apropiados iban siendo reflejados en tablas de arcilla para su uso posterior. La exactitud y objetividad alcanzada por esta transmisión de conocimientos, hizo que el hombre confiase más en los conocimientos escritos que en su memoria.

FIG. 1. Tabla babilónica de 1600 años a. de C.; demuestra el cálculo de la diagonal del cuadrado a partir de los lados, usando el teorema de Pitágoras mil años antes de nacer éste. Los números de la tabla están expresados en base 60; restos de este uso son nuestras actuales divisiones horarias en minutos y segundos.

Hacia el 1600 a. de C., los egipcios comenzaron a usar el papiro para el registro de ciertos contratos mercantiles; el papiro y más adelante el pergamino, fueron los principales medios de registro en Egipto, Persia, Grecia y Roma hasta el año 750 de nuestra era.

Se fundaron las primeras bibliotecas donde se almacenaba la información existente: en Nínive, la biblioteca de Assurbanipal contaba con más de 10 000 tablas de arcilla y, posteriormente, la biblioteca de Alejandría, fundada en el siglo IV a. de C. por iniciativa de Tolomeo I, sobre la que se han hecho cálculos diversos que oscilan en señalar entre 200 000 y 700 000 los papiros que contenía; rival de ésta, fue la biblioteca de Pérgamo, fundada en el siglo II a. de C., que usaba pergamino en sus libros.

1.4. EL ABACO

Hacia el año 1000 a. de C. se inventa el ábaco, que en su forma más simple consiste en una caja llena de arena en donde se van haciendo surcos, uno por cada unidad contada; siguiendo una línea se van haciendo tantos surcos como unidades haya que representar; cuando completan todas las unidades se borran estos surcos y se hace uno siguiendo una línea inmediatamente posterior, y así sucesivamente. En esencia, es un dibujo de un ábaco de cuentas.

Los primeros ábacos de cuentas ensartadas se basaban en el sistema

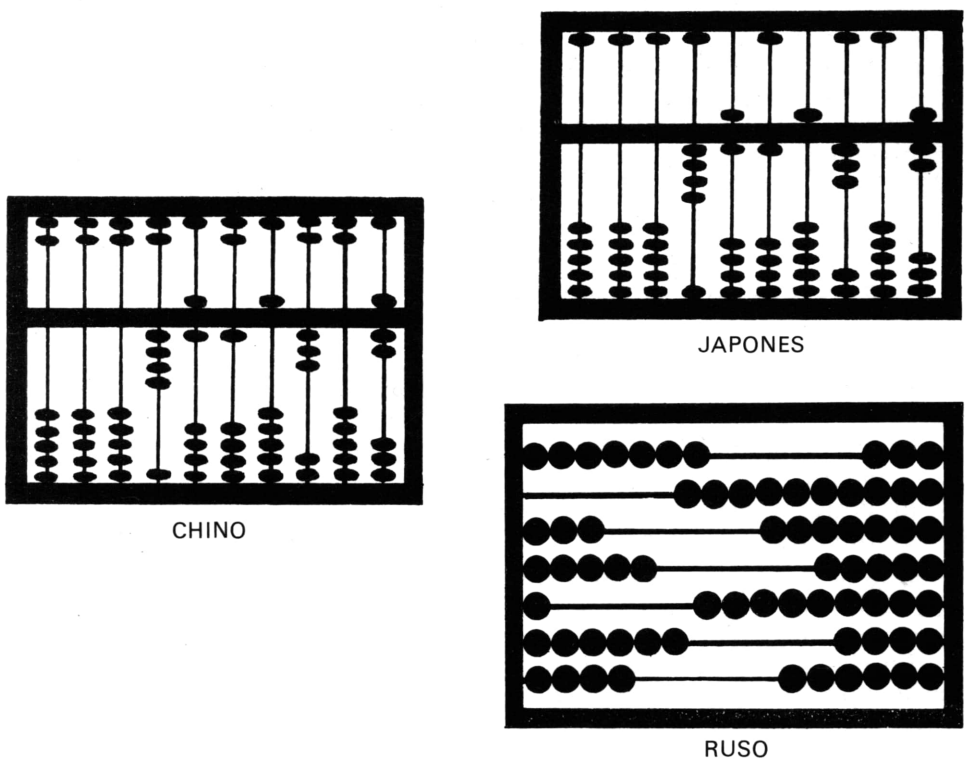


FIG. 2. Abacos chino, japonés y ruso. Todos representan el número 4 6 1 5 3 0 7.

quinario; constaban de alambres con cinco perlas cada uno; al contar, se deplazaban las perlas del primer alambre y cuando todas ellas eran desplazadas se volvían a la posición original desplazando una del alambre siguiente.

Desde la fecha aproximada del descubrimiento del ábaco (posiblemente descubierto independientemente en varias civilizaciones) se fue perfeccionando para ampliar sus aplicaciones e incluso, hacia el año 1946, llegaron a celebrarse diversas competiciones entre ábacos y las primeras máquinas de calcular, no siempre con éxito para estas últimas en cuanto a velocidad y exactitud.

1.5. EL PAPEL Y LA IMPRENTA

En el año 105 de nuestra era, mientras en los países mediterráneos se usaba el papiro y el pergamino, los chinos manufacturaban el papel; sin embargo, hasta el año 750, los árabes y persas no aprenderían su tratamiento y uso; no obstante, fue en el siglo XII cuando los árabes introdujeron en España su manufactura.

A través de España llegó a toda Europa, donde serviría de soporte básico para que los monjes pudieran registrar y transmitir los conocimientos de la época. La tarea de traducir y transcribir a mano era tan costosa como incómoda; además, la creciente demanda de libros, unida a la escasez de personas cualificadas para sus reproducciones fue desarrollando la creatividad de los inventores hasta llegar a su apogeo en el siglo XV.

En 1440, JOHANN GUTEMBERG, presionado por necesidades económicas, mecanizó la producción de libros, inventando así la imprenta.

Significó el comienzo de la educación masiva y el libro impreso se convirtió en el medio de información más eficaz.

1.6. EL SISTEMA DECIMAL

Hacia el año 1120 de nuestra era introducen los árabes en Occidente el sistema decimal, cuyos fundamentos son:

a) Un número se representa por un conjunto de símbolos llamados dígitos.
b) Posición relativa de los dígitos.
c) Introducción del cero.

Este hecho facilitó enormemente los cálculos por la notación tan simple que exige (obsérvese que en las escuelas enseñan números romanos, pero que en ningún momento se insinúa cómo habría que multiplicar dos de ellos).

En el siglo XIII muchos mercaderes italianos usaban el sistema arábigo, hasta el punto que, en 1299, se publicaba un edicto en Florencia prohibiendo a los banqueros usar los símbolos de los infieles.

LEONARDO DA VINCI fue un arduo defensor del sistema decimal.

Fuera de Italia el sistema se extendió con más lentitud, pero hacia 1400 era de uso general en toda Europa.

La introducción de la coma decimal, que separa la parte entera y la parte fraccionaria de un número, se atribuye a PELAZZI DE NIZA, hacia 1492; en 1585 se introduce la primera notación de fracciones.

En el siglo XVII, hacia 1617, NEPER hace la primera introducción a los logaritmos y la iniciación a la regla de cálculo.

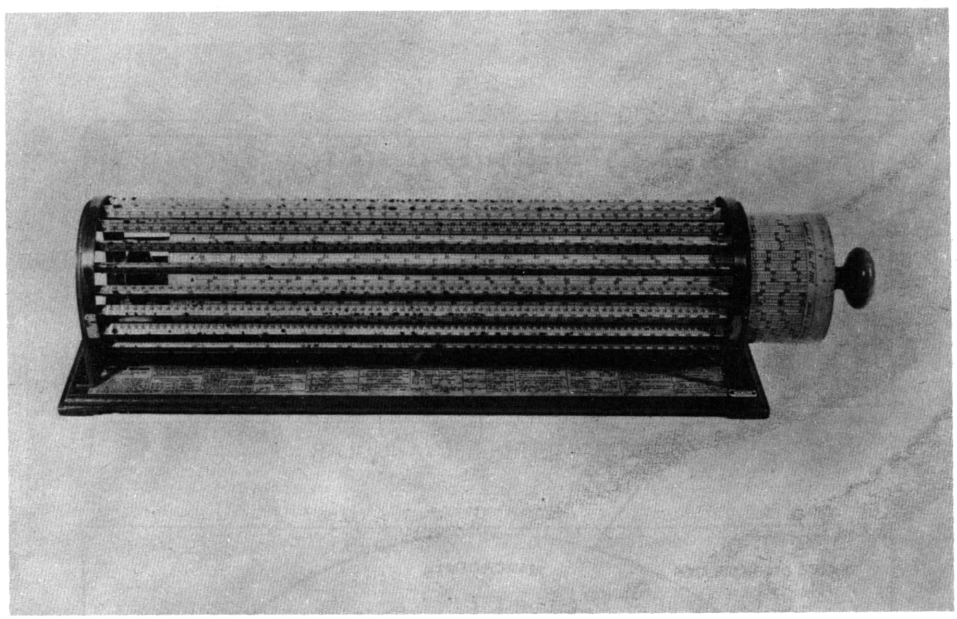


FIG. 3. Regla deslizante de Thacher (1881).

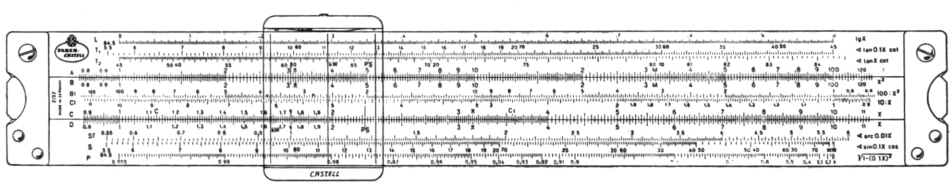
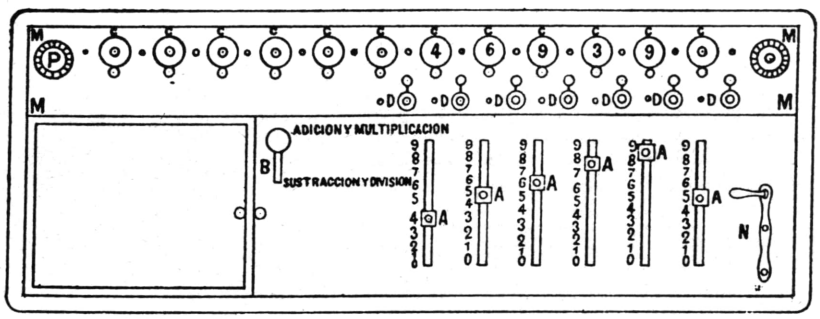


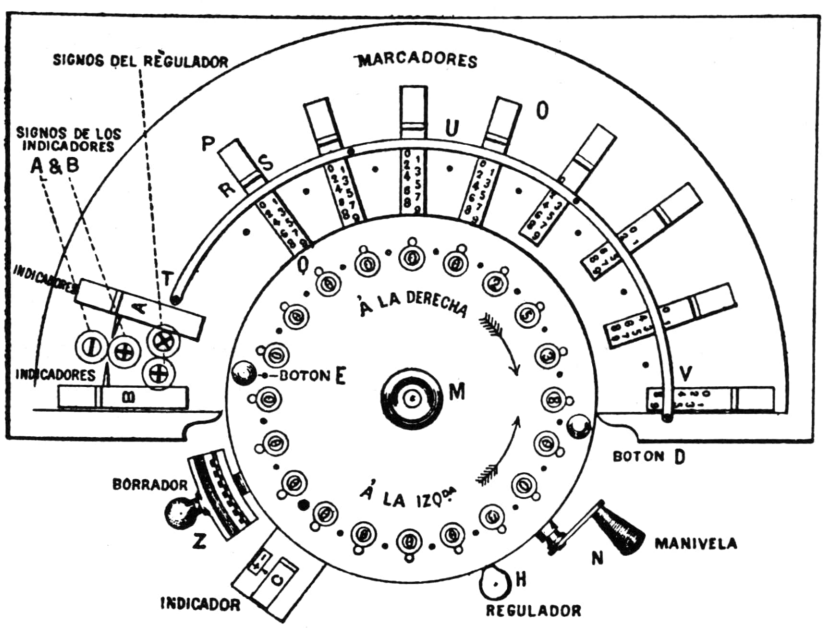
FIG. 4. Regla de cálculo reciente.

1.7. EL ARITMOMETRO DE PASCAL Y EL CALCULADOR DE LEIBNITZ

PASCAL, entre 1623 y 1632, construye la primera sumadora mecánica de seis columnas; consta de seis ruedas dentadas con diez posiciones cada una; cada vez que una rueda da una vuelta completa, mueve mecánicamente una posición de la siguiente. Las multiplicaciones se efectuaban haciendo sumas reiteradas.



ARITMOMETRO RECTANGULAR



ARITMOMETRO CIRCULAR

FIG. 5. Dos versiones de aritmómetros.

El siguiente paso se debe a LEIBNITZ que hace la siguiente descripción de su máquina: «La máquina tiene dos partes, una diseñada para la suma (o resta) y otra para la multiplicación (o división), y las dos se ensamblan; el mecanismo de sumar coincide exactamente con la caja de calcular de Pascal; sin embargo, hay que añadir algo respecto a la multiplicación...» LEIBNITZ estudia el sistema binario, fundamento de los actuales ordenadores.

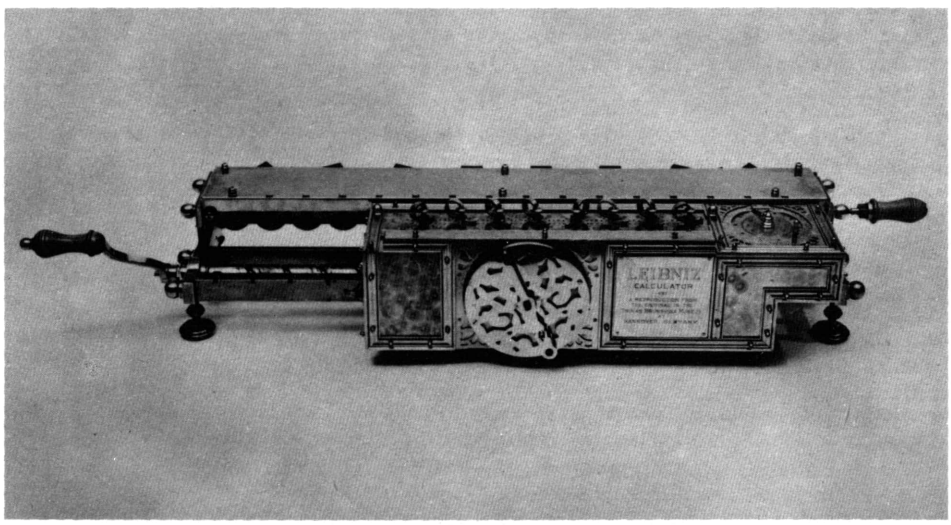


FIG. 6. Calculador de Leibnitz.

Las máquinas de PASCAL y LEIBNITZ eran mecánicas; se trataba ahora de perfeccionar estos mecanismos y a la vez introducir ciertos automatismos que no exijan la acción constante del hombre sobre la máquina.

1.8. LA FICHA PERFORADA

En 1728 se desarrolla en Francia el concepto de ficha perforada aplicada al control de telares y así incrementar la productividad en la industria textil.

Un gran avance en esta línea lo da JAQWARD, en 1801, que inventa un telar capaz de tejer un dibujo previamente perforado en un cartón.

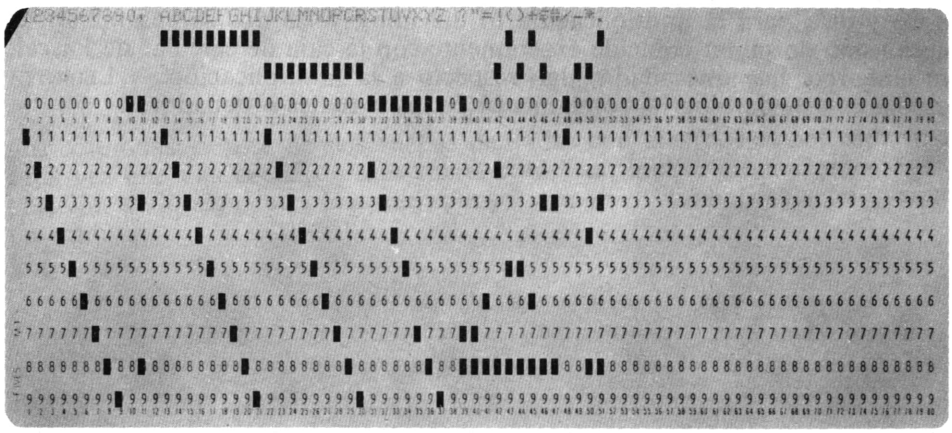


FIG. 7. Ficha perforada.

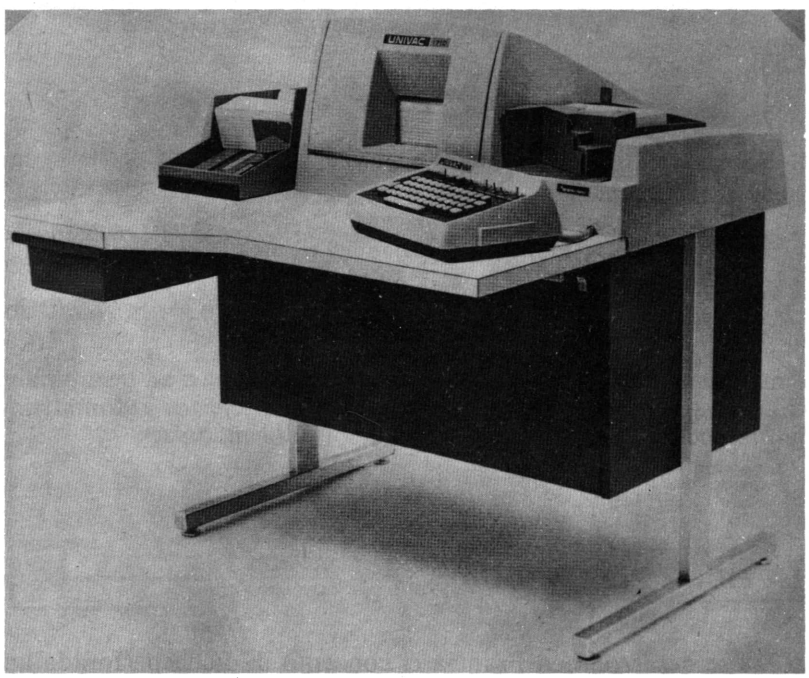


FIG. 8. Perforadora de fichas.

El equivalente a esta máquina, pero sustituyendo el cartón por un rodillo con cabezas de alfileres, es la base del funcionamiento de los organillos.

1.9. LA MAQUINA ANALITICA DE BABBAGE

En 1833, CHARLES BABBAGE diseña una máquina automática, que por falta de presupuesto no llega a construir; la máquina constaba de un mecanismo de cálculo para las cuatro operaciones elementales; una unidad de entrada que mediante fichas perforadas permitía introducir datos e instrucciones, con una memoria capaz de almacenar hasta 1000 datos; una unidad de control que interpreta las instrucciones y las ejecuta, y una unidad de salida para interpretar los resultados del proceso.

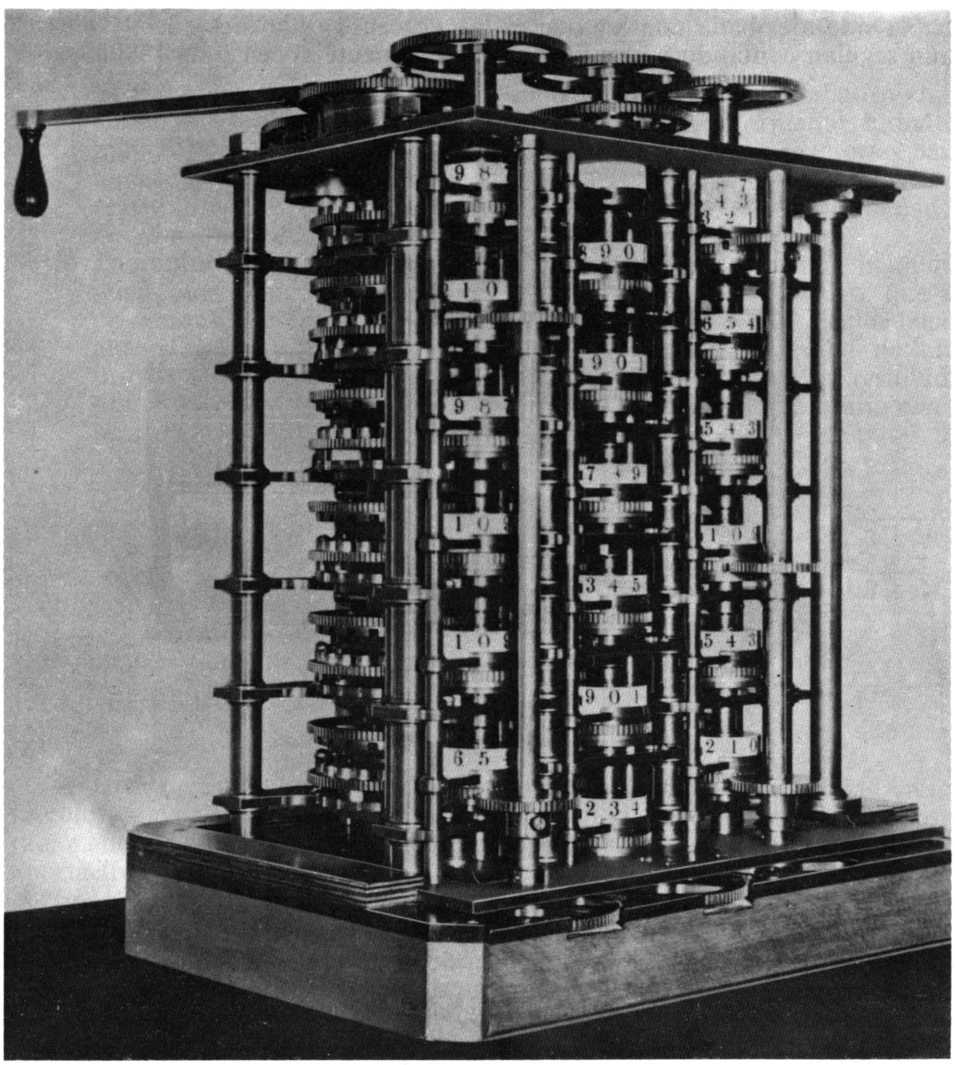


FIG. 9. Máquina analítica de Babbage.

Años más tarde se construyó la máquina y la usó el gobierno inglés para pareparar las tablas de mortalidad que fueron publicadas en 1864. Actualmente se conserva en el museo Británico, de Londres.

1.10. LA TABULADORA DE HOLLERITH

En 1890, HERMAN HOLLERITH desarrolló un ingenio eléctrico para tabular datos estadísticos e inventó un sistema para leer y tabular la información procedente de fichas perforadas.

Este ingenio fue usado para realizar el censo de los Estados Unidos, pues había sido aprobada una ley que exigía un censo cada diez años, y en 1886 aún seguían contando y clasificando manualmente el censo de 1880.

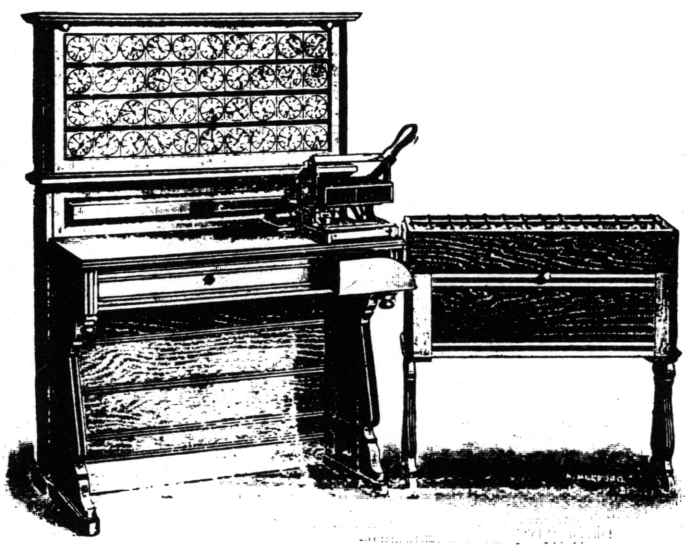


FIG. 10. Tabuladora de Hollerith.

1.11. EL MARK 1

Hemos llegado así al siglo XX; en 1920, IBM construye diversos tipos de calculadoras de oficina y usa la técnica de la tarjeta perforada en algunos de sus modelos. En 1940, el uso de equipos de fichas perforadas estaba bastante extendido.

De 1940 a 1946 se construyen las primeras calculadoras de relés que basan su funcionamiento en el sistema binario; cada número será representado por una cadena de dígitos 0 y 1, y el dígito 0 ó 1 estará indicado por la posición

del relé, activado o no. Siguiendo este procedimiento, HOWARD AIKEN, en 1944, y con la ayuda de IBM, terminó el primer ordenador digital automático a gran escala al que llamó Mark 1. Se trataba de una máquina de cinco toneladas, con miles de relés y tubos de vacío.

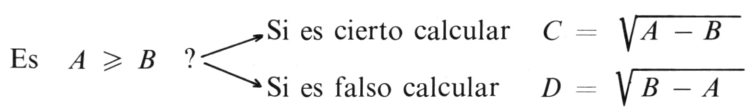
Podemos decir que este ordenador significó el nacimiento de la era de los ordenadores electrónicos.

1.12. ALMACENAMIENTO DE PROGRAMAS: VON NEWMAN

Es VON NEWMAN quien, hacia 1945, promueve el paso decisivo hacia la mecanización del tratamiento digital de la información, con la invención de dos nuevos conceptos:

a) El programa registrado: las máquinas de relés que se usaban podían conservar resultados parciales para utilizarlos posteriormente. El matemático VON NEWMAN tuvo la idea de usar estas mismas memorias para almacenar el programa, de ahí el nombre de «máquina de programa registrado» que recibió este nuevo calculador. Hoy todos los ordenadores funcionan así.

b) La ruptura de secuencia: hasta este momento, cuando una máquina tenía que tomar una decisión lógica, se paraba y esperaba que el operador le diese instrucciones. VON NEWMAN concibió la idea de hacer automáticas las decisiones lógicas, dotando a la máquina de una instrucción llamada salto condicional y según el valor de un resultado obtenido, positivo o negativo, por ejemplo, la máquina ejecutaría una u otra parte del programa.



1.13. EL ENIAC

A partir de este momento, sentadas las bases de rapidez, seguridad en los cálculos, mayor almacenamiento, sin que por ello la máquina crezca desmesuradamente de tamaño, mayor rendimiento, etc., es la ciencia física la que hará aportaciones extraordinarias.

En 1946, aparece el primer ordenador electrónico digital, creado por ECKERT y MAUCHLY, que elimina el uso de piezas mecánicas sustituyéndolas por circuitos formados por tubos de vacío; se trata del ENIAC (Electronic Numerical Integrator and Calculator).

Constaba de unas 18 000 válvulas electrónicas, con un consumo de 174 Kw; pesaba 30 ton., y para un buen rendimiento necesitaba una instalación climatizada cuyo consumo superaba al del propio ordenador. Se estimaba un tiempo de trabajo sin errores del 45 por 100 (el resto para mantenimiento).

El tiempo empleado en realizar una operación elemental era del orden de 1 mseg, es decir, unas mil operaciones por segundo.

1.14. GENERACIONES DE ORDENADORES

Tal como dijimos anteriormente, la ciencia física hará que los diversos componentes del ordenador sean sustituidos progresivamente por otros más evolucionados, dando lugar, de esta forma, a varias generaciones de ordenadores.

a) *Primera generación*

El ENIAC es el ordenador más representativo de esta generación.

b) *Segunda generación*

Comienza a apartir de 1957; las válvulas electrónicas son sustituidas por transistores, que son pequeños paralelepípedos de silicio con base cuadrada, de algunas décimas de milímetro de lado y unas 150 micras de altura; éstos eran montados juntos con los otros componentes (diodos, resistencias, ...) sobre unas galletas del tamaño de una tarjeta postal.

El tiempo de una operación elemental pasó a ser de 100 μseg, es decir, 0,0001 seg, lo que implica la ejecución de unas diez mil operaciones por segundo.

A esta generación pertenece el Siemens 2002.

Lo que en la primera generación significaba usar enormes armarios de válvulas electrónicas, pasó a ser pequeñas cajas de placas transistorizadas en esta segunda generación.

c) *Tercera generación*

Aproximadamente a partir de 1964, los componentes son circuitos integrados; su tamaño es del mismo orden de magnitud que el transistor, pero un circuito integrado puede contener centenas de componentes elementales, en una diminuta ampolla de vidrio, cada una haciendo la misión de un transistor; es decir, el tamaño es aproximadamente del orden de 1/100 de los anteriores.

El tiempo para una operación elemental se estima en 1 μseg, es decir, 10^{-6} seg; esto significa del orden de un millón de operaciones por segundo.

d) *Cuarta generación*

En la que nos encontramos actualmente; los componentes son circuitos integrados a gran escala, de manera que en una ampolla de vidrio hay almacenados millares de componentes elementales.

Actualmente se fabrican placas impresas con centenares de componentes elementales y que pasan perfectamente por el ojo de una aguja de coser.

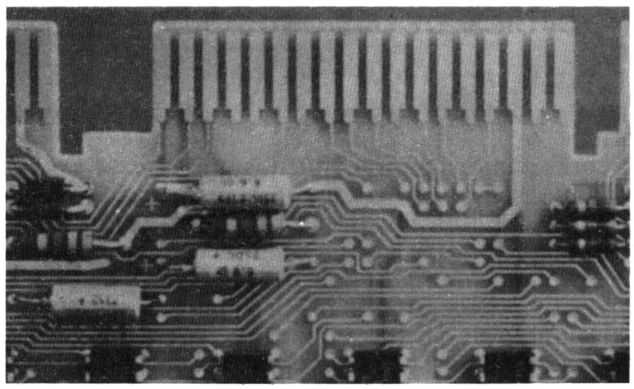


FIG. 11. Placa de un circuito impreso.

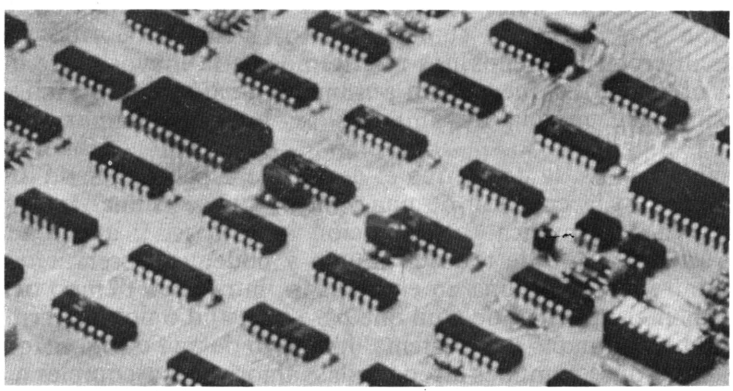
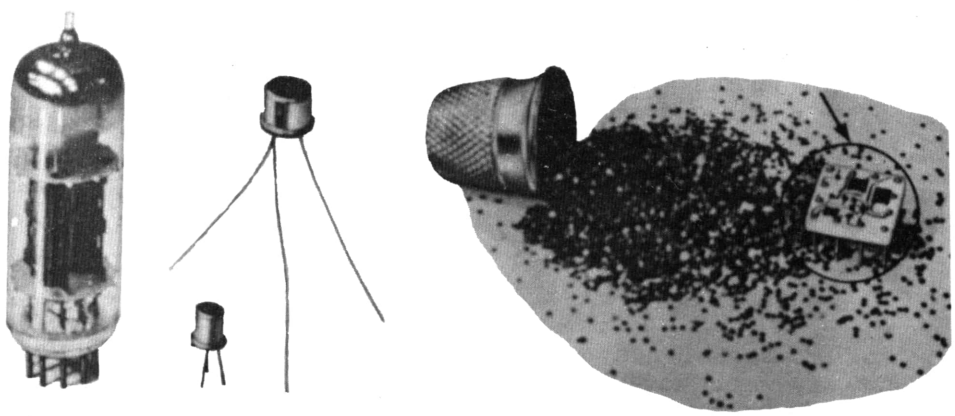


FIG. 12. Circuito integrado.



VALVULA DE VACIO

TRANSISTORES

TRANSISTORES ACTUALES

FIG. 13. Sucesivos componentes elementales.

1.15. CLASIFICACION DE LOS ORDENADORES

Los ordenadores hoy día pueden clasificarse en tres tipos: analógicos, digitales e híbridos.

Lor ordenadores analógicos son aquellos que utilizan como datos magnitudes físicas continuas transformándolas en otra magnitud; por ejemplo, el electrocardiógrafo al recibir los impulsos del corazón los transforma en impulsos eléctricos y éstos a su vez, pueden provocar los desplazamientos de un trazador que dibujará una gráfica.

Otros ejemplos de ordenadores analógicos pueden ser, el cuentakilómetros de un coche (el número de vueltas de una rueda hace girar el contador), un reloj, la regla de cálculo (transforma el desplazamiento continuo de la reglilla en una operación aritmética).

Normalmente trabajan en tiempo real, es decir, a medida que los datos se van produciendo, y sirven para una sola aplicación.

Los ordenadores digitales suelen utilizar el sistema numérico, eso es, la interpretación por cifras de cualquier dato; trabajan, por lo tanto, con magnitudes discretas, y a ellos nos referiremos en la mayor parte del texto.

Existen ordenadores que no son puramente analógicos ni exclusivamente digitales, sino una mezcla de ambos y se les suele llamar ordenadores híbridos.

Finalmente hemos de decir, que el desarrollo espectacular de los ordenadores ha motivado la creación de una ciencia que se ocupa de su estudio y aplicaciones: la Informática (INFORmación autoMÁTICA); a ella se dedican equipos de investigación de todo el mundo y podemos observar que apenas conocemos tal o cual máquina en el mercado, muy poco tiempo después, ha sido ampliamente superada; de aquí que para un estudio no muy profundo del tema, no interese ahondar en el sistema operativo de una u otra máquina, sino en algunas cuestiones comunes a todas ellas, algoritmos asequibles a su forma de trabajo y en algún lenguaje de uso generalizado.

2

Sistema de numeración binario, octal y hexadecimal

2.1. SISTEMA DE NUMERACIÓN BASE n

Supongamos que queremos contar un cajón de bolígrafos a la vez que dejarlos ordenados; comenzamos juntándolos por docenas y supongamos que nos quedan 7 bólígrafos sueltos; las docenas las metemos en cajas de doce cada una y supongamos que conseguimos formar 6 cajas y que nos quedan 9 docenas sueltas.

En total tendremos:

6 cajas, 9 docenas y 7 bolígrafos,

o sea, 697 bolígrafos si convenimos en que la última cifra representa unidades simples, la anterior docenas y la anterior cajas.

El número 697 está expresado en el sistema de base 12.

En general, para representar un número N de objetos, se reúnen éstos por paquetes de n (los objetos sobrantes serán un número inferior a n que lo llamaremos a_0); estos paquetes se agrupan a su vez por paquetes de n (sobrando ahora un número de paquetes menor que n que lo llamaremos a_1), y así sucesivamente. La formulación matemática correspondiente es la siguiente:

$$N = a_p n^p + a_{p-1} n^{p-1} + \cdots + a_1 n + a_0 \qquad [1]$$

donde a_0 es el número de objetos (menor que n) que no pueden ser clasificados en un paquete de n objetos; a_1 es el número de paquetes (menor que n) de n objetos, que no pueden ser clasificados en un paquete de n^2 objetos, etc.

Se dicen que n es la base del sistema de numeración; es de observar que son necesarios $n - 1$ símbolos para poder hacer cualquier representación. Se le añade un n-símbolo 0, cuyo papel es fundamental: si no existe ningún grupo de n^i objetos, en lugar de omitir el término correspondiente en la fórmula [1]|

se le hace figurar con $a_i = 0$. Como todos los términos a_i están representados, puede transcribirse el número N a la forma:

$$N = a_p a_{p-1} \cdots a_1 a_{0(n}$$

Habida cuenta de su construcción, esta representación de N es única. La posición i ocupada por el símbolo a_i indica el número de paquetes de n^i objetos.

Siendo el sistema decimal el sistema de numeración de base 10, a_0 representaría la cifra de las unidades, a_1 la de las decenas, etc.

La elección del sistema decimal parece arbitraria y algunos pretenden abandonarlo en favor del sistema duodecimal, o de base 12, ya que el 12 tiene más divisores que el 10 y algunas operaciones resultarían más simples.

En resumen, si el número N nos lo dan en base n, para pasarlo a base 10 nos basta sustituir n en la fórmula [1], mientras que si N lo tenemos en base 10 y deseamos pasarlo a base n nos basta hacer las agrupaciones anteriores, que no son otra cosa que el cálculo de los restos sucesivos en las divisiones:

$$\begin{array}{llll}
N & \underline{|\,n\quad} & & \\
a_0 & P_1 & \underline{|\,n\quad} & \\
 & a_1 & P_2 & \underline{|\,n\quad} \\
 & & a_2 & P_3
\end{array}$$

$$\begin{array}{ll}
P_{p-1} & \underline{|\,n\quad} \\
a_{p-1} & a_p
\end{array}$$

y N lo expresaremos por:

$$N = a_p a_{p-1} \cdots {}_1 a_{0(n}$$

el paréntesis con la letra n detrás significa que el número está expresado en base n; para los números expresados en base decimal no pondremos esta indicación sobreentendiendo que se trataría de (10.

EJEMPLO 1

Representar en el sistema de base 8 el número 427. Haciendo las divisiones sucesivas se tiene:

$$\begin{array}{lll}
427 & \underline{|\,8\quad} & \\
27 & 53 & \underline{|\,8\quad} \\
3 & 5 & 6
\end{array}$$

Se tiene así:

$$427 = 653_{(8}$$

EJEMPLO 2

Dado en número $2756_{(8}$ *en el sistema de base 8 según se indica, vamos a pasarlo al sistema decimal.*

Aplicamos la fórmula [1] y tenemos:

$$2756_{(8} = 2 \cdot 8^3 + 7 \cdot 8^2 + 5 \cdot 8 + 6 = 1518$$

Observando esto último, vemos que lo realizado es equivalente a calcular el valor numérico del polinomio $2x^3 + 7x^2 + 5x + 6$ para $x = 8$, lo que, como sabemos, puede hacerse aplicando la regla de Ruffini:

	2	7	5	6
8		16	184	1512
	2	23	189	1518

el valo numérico era el último término obtenido, es decir, 1518 tal como nos había salido antes.

En general, si tenemos el número $a_p a_{p-1} \ldots a_1 a_{0(n}$, su valor en el sistema decimal viene dado por la fórmula [1] que, como antes, es el valor numérico del polinomio:

$$a_p x^p + a_{p-1} x^{p-1} + \cdots + a_1 x + a_0$$

en $x = n$, que por la regla de Ruffini es el valor de b_p dado por:

	a_0	a_1	a_2	...	a_p
n		nb_0	nb_1	...	nb_{p-1}
	b_0	b_1	b_2	...	b_p

EJEMPLO 3

Si en el sistema de base 12 indicamos el número 10 por a *y el 11 por* b, *expresar el número 7895 en base 12.*

Por divisiones sucesivas:

7895	12		
069	657	12	
095	57	54	12
11	*9*	*6*	*4*

luego:

$$7895 = 469b_{(12}$$

EJEMPLO 4

Expresar el número 327 en el sistema de base 2.

327	| 2							
12	163	| 2						
07	03	81	| 2					
1	*1*	*01*	40	| 2				
			00	20	| 2			
				0	10	| 2		
					0	5	| 2	
						1	2	| 2
							0	*1*

así:

$$327 = 101000111_{(2}$$

EJEMPLO 5

El número $1011101_{(2}$ expresarlo:

a) En el sistema decimal.
b) En el sistema de base 8.

a) Por Ruffini:

	1	0	1	1	1	0	1
2		2	4	10	22	46	92
	1	2	5	11	23	46	93

y

$$1011101_{(2} = 93$$

b) Pasamos ahora el número 93 a base 8:

93	| 8		
13	11	| 8	
5	*3*	*1*	

$$1011101_{(2} = 135_{(8}$$

EJEMPLO 6

Expresar el número $2357_{(8}$ en el sistema de base 2.

Lo pasaremos en primer lugar a base 10:

	2	3	5	7
8		16	152	1256
	2	19	157	1263

así pues,

$$2357_{(8} = 1263$$

pasamos ahora 1263 a base 2 y tenemos:

1263	|2									
06	631	|2								
03	03	315	|2							
1	11	11	157	|2						
	1	15	17	78	|2					
		1	*1*	18	39	|2				
				0	19	19	|2			
					1	*1*	9	|2		
							1	4	|2	
								0	2	|2
									0	*1*

$$1263 = 10011101111_{(2} = 2357_{(8}$$

2.2. OPERACIONES CON NUMEROS EN BASE n

No pretendemos en este apartado hacer un estudio exhaustivo de las operaciones con números en una base cualquiera, sino tan sólo hacer una breve recordatoria práctica de las dos operaciones más sencillas, la adición y la multiplicación, que nos permitan comprender con mayor profundidad el siguiente apartado.

Según sabemos, la adición y multiplicación de números expresados en base decimal, viene definida a partir de las correspondientes tablas de sumar y de multiplicar entre números de una sola cifra.

Para las operaciones con números en una base n tenemos dos posibles caminos:

a) Pasar los números a base decimal, efectuar las operaciones pedidas y el resultado pasarlo a la base n de origen.
b) Construir las correspondientes tablas de sumar y multiplicar en base n y a partir de ahí efectuar las operaciones en base n siguiendo reglas análogas a las operaciones en base 10.

EJEMPLO 1

Efectuar de las dos formas anteriormente indicadas:

$$234_{(7} + 23_{(7} \qquad \text{y} \qquad 234_{(7} \times 23_{(7}$$

a) Teniendo en cuenta que:

$$234_{(7} = 2 \cdot 7^2 + 3 \cdot 7 + 4 = 123$$
$$23_{(7} = 2 \cdot 7 + 3 = 17$$

se tiene

$$234_{(7} + 23_{(7} = 123 + 17 = 140 = 260_{(7}$$
$$234_{(7} \times 23_{(7} = 123 \times 17 = 2091 = 6045_{(7}$$

b) Construyamos las tablas de sumar y multiplicar en base 7, son:

la de sumar:

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	10
2	2	3	4	5	6	10	11
3	3	4	5	6	10	11	12
4	4	5	6	10	11	12	13
5	5	6	10	11	12	13	14
6	6	10	11	12	13	14	15

análogamente la de multiplicar:

×	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	11	13	15
3	0	3	6	12	15	21	24
4	0	4	11	15	22	26	33
5	0	5	13	21	26	34	42
6	0	6	15	24	33	42	51

con estas tablas, y teniendo en cuenta el arrastre tal como en el sistema decimal, podemos efectuar cualquier multiplicación o suma en base 7. En nuestro caso se tiene:

$$\begin{array}{r} 234 \\ +\ 23 \\ \hline 260_{(7} \end{array} \qquad \begin{array}{r} 234 \\ \times\ 23 \\ \hline 1035 \\ 501 \\ \hline 6045_{(7} \end{array}$$

EJEMPLO 2

Calcular las siguientes sumas y productos en base 7 a partir de las tablas:

$32_{(7} \cdot 5 =$
$131_{(7} \cdot 10_{(7} =$
$243_{(7} + 36_{(7} =$
$425_{(7} + 14_{(7} + 361_{(7} =$

los resultados serían:

$$\begin{array}{r} 32 \\ \times\ 5 \\ \hline 223_{(7} \end{array} \qquad \begin{array}{r} 131 \\ \times\ 10 \\ \hline 1310_{(7} \end{array} \qquad \begin{array}{r} 243 \\ +\ 36 \\ \hline 312_{(7} \end{array} \qquad \begin{array}{r} 425 \\ 14 \\ 361 \\ \hline 1133_{(7} \end{array}$$

EJEMPLO 3

Construir las tablas de sumar y multiplicar en base 2 y aplicarlas al cálculo de:

$1011_{(2} \cdot 11_{(2} =$
$10110_{(2} + 111_{(2} + 1000_{(2} =$

Las tablas son:

+	0	1
0	0	1
1	1	10

×	0	1
0	0	0
1	0	1

se tiene entonces que:

$$\begin{array}{r} 1011 \\ \times\ 11 \\ \hline 1011 \\ 1011 \\ \hline 100001_{(2} \end{array} \qquad \begin{array}{r} 10110 \\ 111 \\ 1000 \\ \hline 100101_{(2} \end{array}$$

EJEMPLO 4

Igual que el ejemplo 3 para base 5 y calcular:

$234_{(5} + 12_{(5} + 1434_{(5} =$
$2322_{(5} + 120_{(5} + 4444_{(5} =$
$132_{(5} \cdot 43_{(5} =$
$4321_{(5} \cdot 4 =$

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	10
2	2	3	4	10	11
3	3	4	10	11	12
4	4	10	11	12	13

×	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	11	13
3	0	3	11	14	22
4	0	4	13	22	31

los cálculos a partir de las tablas son inmediatos:

234	2322	132	4321
12	120	43	4
1434	4444	1001	$33334_{(5}$
$2240_{(5}$	$12441_{(5}$	1133	
		$12331_{(5}$	

2.3. EL SISTEMA BINARIO

Como hemos visto, consta de los símbolos 0 y 1; así como es muy difícil encontrar en la naturaleza entes con diez aspectos completamente diferenciados, resulta muy fácil encontrar fenómenos con sólo dos aspectos diferentes que puedan representar los símbolos 0 y 1; por ejemplo, una bombilla o está apagada o está encendida, en un circuito o pasa o no pasa corriente por él, un núcleo de ferrita está imantado o no, etc.

El sistema binario fue introducido por LEIBNITZ en el siglo XVII y es el más adecuado a las máquinas electrónicas, ya que éstas utilizan dos estados estables (esto se verá con más detalle más adelante, cuando hablemos de las memorias).

La representación de los primeros dígitos en binario es:

DECIMAL	BINARIO
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

Ya hemos visto en el apartado 2.1 cómo pasar un número de decimal a binario y recíprocamente.

Un caso que no tratamos entonces es el de si se tiene un número en binario con una parte fraccionaria; por ejemplo, el número

$$10011.1011_{(2}$$

para pasar este tipo de números a base 10 basta tener en cuenta que así como las cifras de la parte entera se multiplican por las potencias positivas de 2 al

pasar el número a base 10, las cifras de la parte fraccionaria se multiplicarán por las potencias negativas de 2, es decir, se tienen las igualdades:

.
.
.

$100_{(2} = 2^3$
$110_{(2} = 2^2$
$10_{(2} = 2^1$
$1 = 2^0$
$0.1_{(2} = 2^{-1}$
$0.01_{(2} = 2^{-2}$
$0.001_{(2} = 2^{-3}$

.
.
.

Así pues, se tiene que

$$10011.\ 1011_{(2} = 2^4 + 2 + 1 + 2^{-1} + 2^{-3} + 2^{-4} = 19 + 11/16 = 19.6875$$

EJEMPLO 2

Expresar en base 10 los siguientes números expresados en binario:

$1.01_{(2} =$
$1010.101_{(2} =$
$1000.001_{(2} =$

según lo dicho anteriormente es:

$$1.01_{(2} = 1 + 2^{-2} = 1 + 1/4 = 1.25$$
$$1010.101_{(2} = 2^3 + 2 + 2^{-1} + 2^{-3} = 10 + 5/8 = 10.625$$
$$1000.001_{(2} = 2^3 + 2^{-3} = 8 + 1/8 = 8.125$$

Vemos el proceso inverso: cómo pasar un número en base 10 con cifras decimales a binario; este número lo podemos dividir en dos partes, una la parte entera, cuyo paso al sistema binario ya sabemos hacer, y otra la parte decimal la que intentaremos pasar al sistema binario.

EJEMPLO 3

Sea el número 437.625; se tiene que:

$$437.625 = 437 + 0.625$$

y que

$$437 = 110110101_{(2}$$

se trata pues, de pasar 0.625 a base 2; para ello supongamos que su representación binaria fuese:

$$0.625 = 0.abcd. \cdots_{(2}$$

lo que es equivalente a escribir:

$$0{,}625 = a/2 + b/2^2 + c/2^3 + \cdots$$

donde a, b, c, $\cdots$ son ó 0 ó 1; si multiplicamos ambos miembros de la igualdad anterir por 2 se tiene:

$$0.625 \cdot 2 = a + b/2 + c/2^2 + \cdots$$

luego, a es la parte entera de $0.625 \cdot 2 = 1.250$, es decir, $a = 1$ quedaría:

$$0.250 = b/2 + c/2^2 + \cdots$$

multiplicamos nuevamente por 2 y se tiene:

$$0.250 \cdot 2 = b + c/2 + \cdots$$

y por consiguiente,

$$b = \text{parte entera de } 0.250 \cdot 2 = \text{parte entera } (0.5) = 0$$

Tendríamos ahora que:

$$0.5 = c/2 + d/2^2 + \cdots$$

y

$$c = \text{parte entera de } 0.5 \cdot 2 = \text{parte entera } (1) = 1$$

nos queda entonces que:

$$0 = d/2 + \cdots$$

y, por consiguiente, todas las demás cifras serían ceros, pues nos hemos quedado sin parte decimal.

Luego:

$$0.625 = 0.101_{(2}$$

Si en el desarrollo del ejemplo anterior ponemos en lugar de 0.625 un número en base 10 de la forma $0.ABC\ \cdots$ llegaríamos a la misma conclusión, es decir, que:

$$0.ABC\ \cdots\ = 0.abc\ \cdots\ _{(2}$$

a = parte entera $(2 \cdot 0.ABC \cdots\cdot)$ = parte entera (N_1)
b = parte entera $2 \cdot (N_1 - a)$ = parte entera (N_2)
c = parte entera $2 \cdot (N_2 - b)$ = parte entera (N_3)

. . . .
. . . .
. . . .

EJEMPLO 4

Vamos a pasar a binario los siguientes números:

a) 0.125; *b*) 35.125; *c*) 19.5.

a)

$0.125 \cdot 2 = 0.250$ parte entera $(0.250) = 0$
$0.250 \cdot 2 = 0.5$ parte entera $(0.5) = 0$
$0.5 \cdot 2 = 1$ parte entera $(1) = 1$

luego:

$$0.125 = 0.001_{(2}$$

b) Como $35.125 = 35 + 0.125$

35 ·	|2				
15	17	|2			
1	*1*	8	|2		
		0	4	|2	
			0	2	|2
				0	*1*

Según esto:

$$35.125 = 35 + 0.125 = 100011_{(2} + 0.001_{(2} = 100011.001_{(2}$$

c) $19.5 = 19 + 0.5$

$$\begin{array}{ccccc}
19 & \underline{|2} & & & \\
\mathit{1} & 9 & \underline{|2} & & \\
 & \mathit{1} & 4 & \underline{|2} & \\
 & & \mathit{0} & 2 & \underline{|2} \\
 & & & \mathit{0} & \mathit{1}
\end{array}$$

$$0.5 \cdot 2 = 1 \quad \text{parte entera } (1) = 1$$

luego

$$19.5 = 10011_{(2} + 0.1_{(2} = 10011.1_{(2}$$

Una fuente de errores en las primeras calculadoras era la aparición de números periódicos en el sistema binario que no lo son en el sistema decimal.

Así, tenemos el caso del número 0.4 que vamos a pasarlo a binario como antes:

$$\begin{aligned}
0.4 \cdot 2 &= 0.8 \quad \text{parte entera } (0.8) = 0 \\
0.8 \cdot 2 &= 1.6 \quad \text{parte entera } (1.6) = 1 \\
0.6 \cdot 2 &= 1.2 \quad \text{parte entera } (1.2) = 1 \\
0.2 \cdot 2 &= 0.4 \quad \text{parte entera } (0.4) = 0
\end{aligned}$$

comienzan a repetirse las cifras, por lo que tenemos:

$$0.4 = 0.0110011001 \cdots\cdot = 0.\widehat{0110}_{(2}$$

EJEMPLO 5

Pasar al sistema binario los siguientes números decimales:

a) 27.8; *b*) 60.3

a)

$$\begin{aligned}
0.8 \cdot 2 &= 1.6 \quad \text{parte entera } (1.6) = 1 \\
0.6 \cdot 2 &= 1.2 \quad \text{parte entera } (1.2) = 1 \\
0.2 \cdot 2 &= 0.4 \quad \text{parte entera } (0.4) = 0 \\
0.4 \cdot 2 &= 0.8 \quad \text{parte entera } (0.8) = 0
\end{aligned}$$

$$0.8 = 0.\widehat{1100}_{(2}$$

pasemos ahora la parte entera:

$$\begin{array}{rrrrrr} 27 & \vert 2 & & & & \\ 17 & 13 & \vert 2 & & & \\ 1 & 1 & 6 & \vert 2 & & \\ & & 0 & 3 & \vert 2 & \\ & & & 1 & 1 & \end{array}$$

$$27 = 11011_{(2}$$

se tiene entonces que:

$$27.8 = 11011.\widehat{1100}_{(2}$$

b) $60.3 = 60 + 0.3$

$$\begin{array}{rrrrrr} 60 & \vert 2 & & & & \\ 0 & 30 & \vert 2 & & & \\ & 10 & 15 & \vert 2 & & \\ & 0 & 1 & 7 & \vert 2 & \\ & & & 1 & 3 & \vert 2 \\ & & & & 1 & 1 \end{array}$$

la parte decimal sería:

$$\begin{aligned} 0.3 \cdot 2 &= 0.6 \quad \text{parte entera } (0.6) = 0 \\ 0.6 \cdot 2 &= 1.2 \quad \text{parte entera } (1.2) = 1 \\ 0.2 \cdot 2 &= 0.4 \quad \text{parte entera } (0.4) = 0 \\ 0.4 \cdot 2 &= 0.8 \quad \text{parte entera } (0.8) = 0 \\ 0.8 \cdot 2 &= 1.6 \quad \text{parte entera } (1.6) = 1 \end{aligned}$$

$$0.3 = 0.\widehat{01001}_{(2}$$

Y

$$60.3 = 111100.\widehat{01001}_{(2}$$

Finalmente, una relación interesante es la que existe entre el número de cifras de un número en el sistema decimal y su número de cifras en el sistema de base dos:

$$2^m - 1 = 10^{m \log 2} - 1$$

esto indica, que un número con m cifras en base dos, tendrá, aproximadamente, $0.3m$ cifras en base 10, lo que significa que el número de dígitos de un

número en base dos es, aproximadamente, tres veces superior al número de dígitos en base 10.

2.4. NUMERACIONES OCTAL Y HEXADECIMAL

Si la representación binaria resulta muy cómoda para el ordenador, no ocurre lo mismo para el usuario, que prefiere una forma más condensada de escritura, es decir, no resulta fácil ver con rapidez la representación decimal del número 10010011111000011, e incluso al ordenador, efectuar la conversión le lleva cierto tiempo.

Tomemos el sistema de numeración de base 8 cuyos dígitos en representación binaria de tres cifras es:

0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

la traducción de estos ocho dígitos es inmediata; pues bien, consideremos el número 287 en base 10; lo pasamos a base 8, lo cual requiere menos laboriosidad que el paso a base dos:

$$287 = 437_{(8} = 4 \cdot 8^2 + 3 \cdot 8 + 7 = 4 \cdot 2^6 + 3 \cdot 2^3 + 7$$

consideremos ahora las divisiones sucesivas de este número por dos para pasarlo a binario:

$$287 = 4 \cdot 2^6 + 3 \cdot 2^3 + 7$$

/2	queda	$4 \cdot 2^5 + 3 \cdot 2^2 + 3$	y resto	1
/2	queda	$4 \cdot 2^4 + 3 \cdot 2 + 1$	y resto	1
/2	queda	$4 \cdot 2^3 + 3$	y resto	1
/2	queda	$4 \cdot 2^2 + 1$	y resto	1
/2	queda	$4 \cdot 2$	y resto	1
/2	queda	4	y resto	0
/2	queda	2	y resto	0
/2	queda	1	y resto	0
/2	queda	0	y resto	1

observemos que los tres primeros restos los aporta el 7; los tres restos siguientes los aporta el 3 y los tres siguientes el 4.

Por lo tanto, para pasar un número de base 8 a base 2, basta con pasar cada cifra a base 2 en representación binaria de tres dígitos.

$$287 = 437_{(8} = \underbrace{1\,0\,0}_{4}\ \underbrace{0\,1\,1}_{3}\ \underbrace{1\,1\,1}_{7}{}_{(2}$$

Análogamente puede hacerse en el sistema de base hexadecimal (16) usando grupos de cuatro dígitos en binario.

La mayoría de los ordenadores efectúan este doble cambio, de decimal a octal (o hexadecimal) y de éste a binario; la elección de un sistema u otro depende de la unidad de almacenamiento que tenga, es decir, de la longitud de las palabras como veremos en capítulos posteriores.

EJEMPLO 2

Pasar a binario pasando por octal, los siguientes números:

a) 273; *b*) 1492; *c*) 795

a)

273	8	
33	34	8
1	*2*	*4*

$$273 = 421_{(8} = \underbrace{100}_{4}\ \underbrace{010}_{2}\ \underbrace{001}_{1}{}_{(2}$$

b)

1492	8		
69	186	8	
52	26	23	8
4	*2*	*7*	*2*

luego se tiene:

$$1492 = 2724_{(8} = \underbrace{101}_{2}\ \underbrace{111}_{7}\ \underbrace{010}_{2}\ \underbrace{100}_{4}$$

c)

795	8		
75	99	8	
3	19	12	8
	3	*4*	*1*

$$795 = 1433_{(8} = \underbrace{1}_{1}\ \underbrace{100}_{4}\ \underbrace{011}_{3}\ \underbrace{011}_{3}$$

EJEMPLO 3

Pasar a octal los siguientes números en binario:

a) 1011101101; *b*) 1000011101; *c*) 10101010

a) $\underbrace{101}_{5}\underbrace{110}_{6}\underbrace{110}_{6}\underbrace{101}_{5} = 1355_{(8}$

b) $\underbrace{100}_{4}\underbrace{001}_{1}\underbrace{110}_{6}\underbrace{101}_{5} = 1035_{(8}$

c) $\underbrace{101}_{5}\underbrace{010}_{2}\underbrace{10}_{2} = 252_{(8}$

2.5. EJERCICIOS

1. Sumar en el sistema de base 7:

$$3640_{(7} + 1652_{(7} + 346_{(7} =$$

2. Sumar en el sistema de base 12:

$$3675a_{(12} + 2b634_{(12} =$$

3. Expresar en el sistema de base 12 el número 34126.

4. Expresar en el sistema de base decimal el número $27294_{(11}$.

5. Hallar n tal que:

$$370_{(9} = 10302_{(n}.$$

6. Hallar la expresión en base 10 del número $0.4532_{(7}$.

7. El número $0.1_{(2}$, expresarlo en base 10.

8. El número $0.01_{(2}$ expresarlo en base decimal.

9. Expresar en base decimal el número $27.234_{(8}$.

10. Hallar n tal que: $127 = 1002_{(n}$.

11. Pasar a base dos los siguientes números decimales:

a) 127.15; *b*) 9.35; *c*) 0.05; *d*) 18.725

12. Pasar a base 10 los siguientes números:

a) $1011.0011_{(2}$; *b*) $101.101_{(2}$; *c*) $11.011_{(2}$; *d*) $10.01_{(2}$.

13. Pasar a base dos pasando por base 8 los siguientes números:

a) 765; *b*) 249; *c*) 2357; *d*) 475

14. Pasar a base 10 pasando por octal los números siguientes:

a) $11011010101_{(2}$; *b*) $10010011110_{(2}$; *c*) $11111011101111_{(2}$

15. Construir las tablas de sumar y de multiplicar en base 8 y efectuar las siguientes operaciones:

a) $6745_{(8} + 276_{(8} + 100_{(8} =$

b) $327_{(8} \cdot 14_{(8} =$

c) $(432_{(8} + 561_{(8}) \cdot 34_{(8} =$

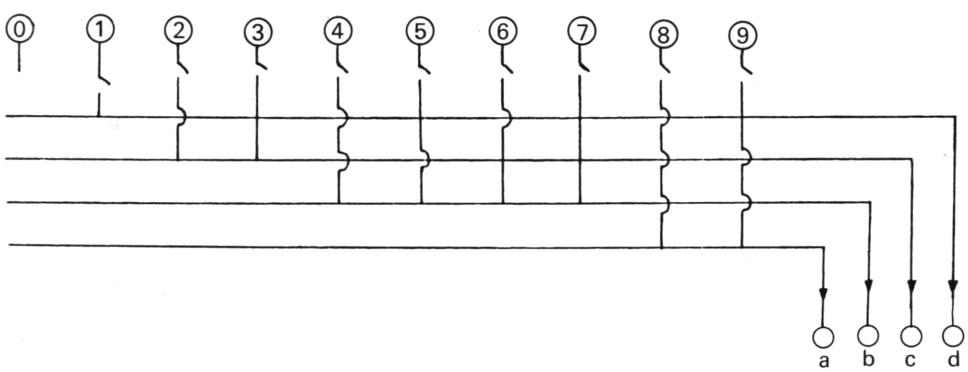
3

El sistema binario en ordenadores

3.1. UN DISEÑO MUY SIMPLE

Adoptamos el convenio de que en un circuito eléctrico una luz encendida, un interruptor cerrado y, en general, el paso de corriente nos indicará un 1; mientras que una luz apagada, un interruptor abierto o algo que indique el no paso de corriente nos indicará un 0. Un interruptor abierto lo representaremos por:

Un esquema de conversión de los dígitos decimales a binario, de construcción muy fácil puede ser el siguiente:

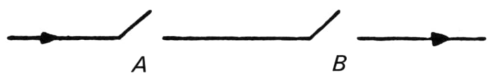


El diseño como puede verse es simbólico, pero lo que se pretende mostrar es que al pulsar la tecla correspondiente a un dígito, pasa corriente por los nodos indicados provocando que se enciendan las luces *a, b, c* o *d* correspondientes. Así, por ejemplo, al pulsar el 5, *a* y *c* permanecen apagadas, mientras que *b* y *c* se encienden, con lo que se tiene el número 0 1 0 1 que es el 5 en binario.

3.2. TEORIA DE CIRCUITOS LOGICOS

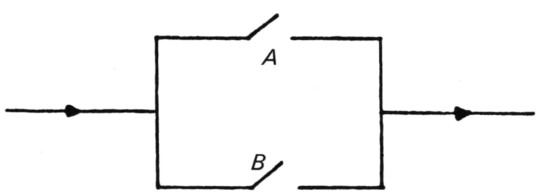
En un ordenador se repiten continuamente tres tipos de circuitos llamados puertas: puerta Y, puerta O y puerta NO.

PUERTA Y: *A* y *B* son dos interruptores; el sentido de la corriente es de izquierda a derecha; lógicamente sólo circulará corriente cuando los interruptores *A* y *B* estén cerrados, es decir, cuando *A* y *B* valgan ambos 1:



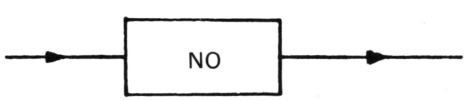
A	*B*	SALIDA
1	1	1
1	0	0
0	1	0
0	0	0

PUERTA O: En este caso, como se ve por la figura, circulará corriente cuando uno de los dos interruptores estén cerrados:



A	*B*	SALIDA
1	1	1
1	0	1
0	1	1
0	0	0

PUERTA NO: Su diseño es más complicado que las anteriores por lo que simplemente la representaremos según indica la figura, y nos limitaremos a exponer su efecto:



ENTRADA	SALIDA
1	0
0	1

es decir, cuando entra un 1 sale un 0, y viceversa.

3.3. ALGEBRA DE BOOLE DE LOS CONMUTADORES

La lógica de los circuitos conmutadores siguen las leyes de un álgebra de Boole con dos conjuntos $\{\varnothing, I\}$, es decir, los conjuntos vacío y universal; el vacío, $\varnothing$, correspondiendo al 0, y el universal, I, correspondiendo al 1. Las puertas Y, O y NO representan la intersección, unión y complementación,

respectivamente, como puede comprobarse comparando con las siguientes tablas:

UNIÓN

A	B	$A \cup B$
I	I	I
I	∅	I
∅	I	I
∅	∅	∅

INTERSECCION

A	B	$A \cap B$
I	I	I
I	∅	∅
∅	I	∅
∅	∅	∅

COMPLEMENTACIÓN

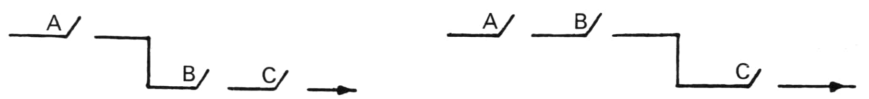
A	A^c
I	∅
∅	I

Por las tablas se deducen las propiedades conmutativa, idempotente, la existencia del elemento neutro ∅ para la unión y I para la intersección y las propiedades de la complementación:

$$A \cup A^c = \mathrm{I} \qquad A \cap A^c = \varnothing$$

Los siguientes circuitos prueban las propiedades indicadas por ser los efectos idénticos:

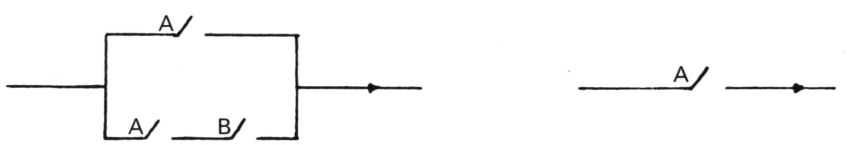
Asociativa:



$$A \cap (B \cap C) = (A \cap B) \cap C$$

$$(A \cup B) \cup C = A \cup (B \cup C)$$

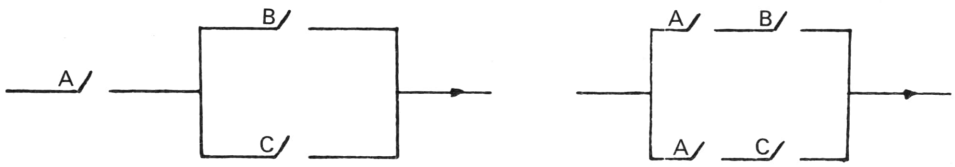
Simplificativa:



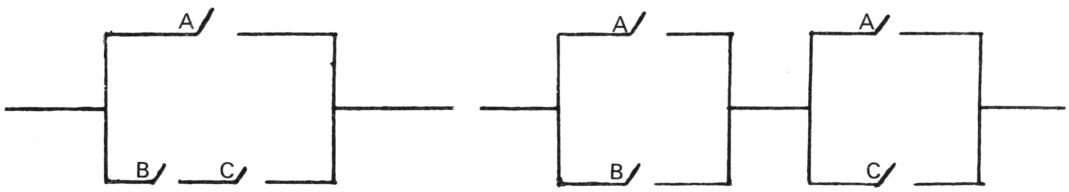
$$A \cup (B \cap A) = A$$

$$A \cap (B \cup A) = A$$

Distributiva:



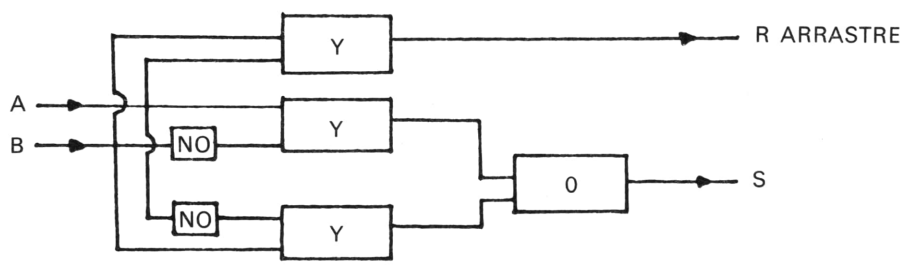
$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$



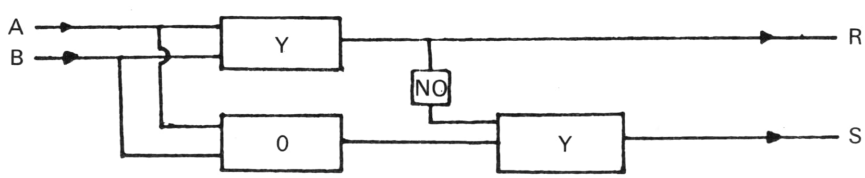
$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

Sabiendo que los circuitos conmutadores forman un álgebra de Boole, los diseños de circuitos pueden simplificarse bastante.

Por ejemplo, un circuito que responde a la suma binaria es:



En este circuito $R = A \cap B$ y $S = (A \cap B^c) \cup (A^c \cap B)$ que aplicándole las leyes del álgebra de Boole se convierte en $S = (A \cap B)^c \cap (A \cup B)$ cuyo diseño es más simple como se ve por la figura:

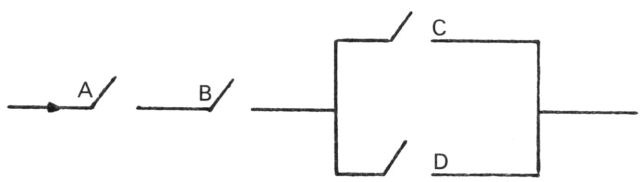


Este circuito nos daría la suma binaria; nótese que el producto binario es la puerta Y simplemente.

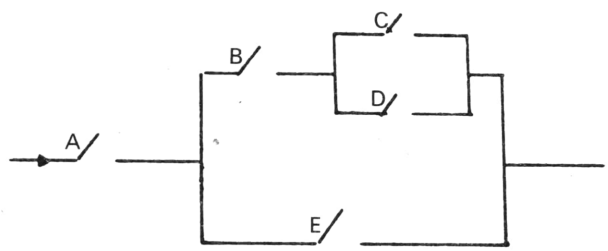
3.4. EJERCICIOS

1. Hallar la expresión booleana de cada uno de los siguientes circuitos:

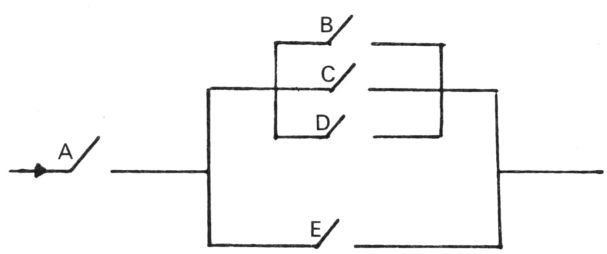
a)



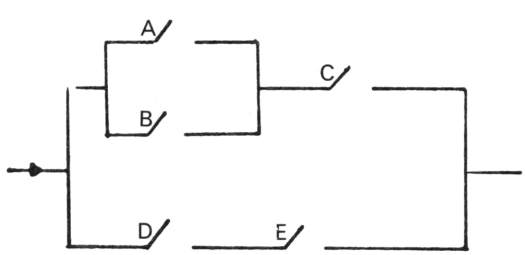
b)



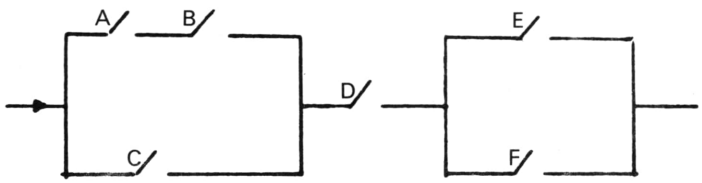
c)



d)



e)



2. Construir un circuito para cada una de las siguientes expresiones booleanas:

a) $A \cap (B \cup C) \cap D$

b) $[A \cap (C \cup D)] \cup B$

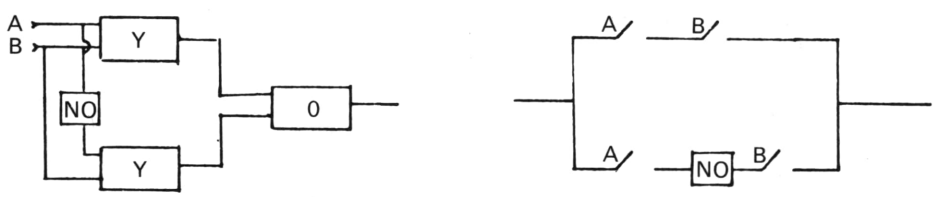
c) $\{[(A \cup B) \cap D] \cup E\} \cap F$

d) $(A \cap B) \cup [C \cap (E \cup F \cup G)]$

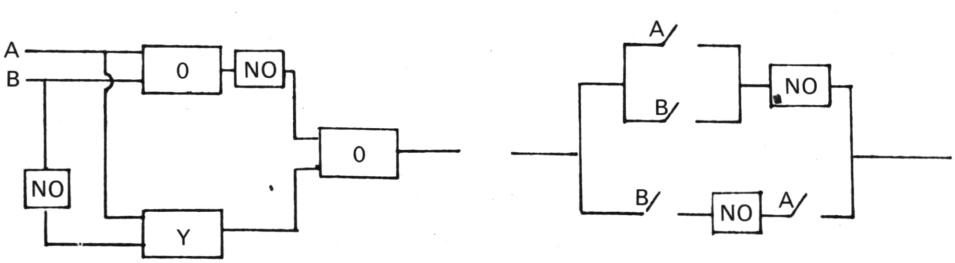
e) $[A \cup (B \cap C)] \cap E$

3. Verificar que en los dos casos siguientes las parejas de circuitos representadas son equivalentes. Hallar las expresiones booleanas de cada uno de ellos.

a)

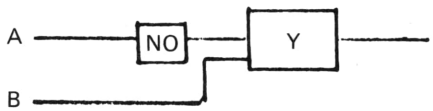


b)

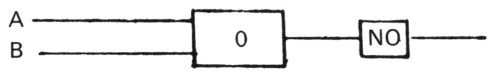


4. Hallar la versión quivalente, con interruptores y puertas NO exclusivamente, así como la expresión booleana de los siguientes circuitos:

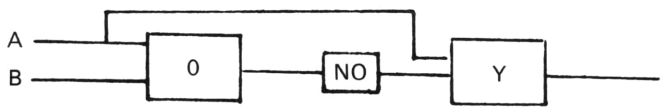
a)



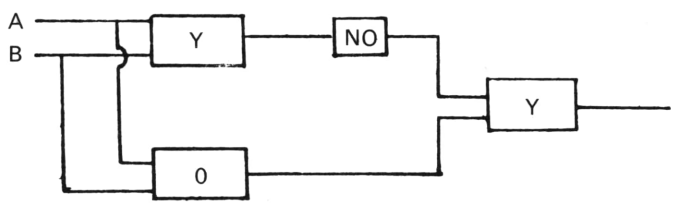
b)



c)



d)



5. Usando las puertas Y, O y NO construir los circuitos correspondientes a las siguientes expresiones booleanas:

a) $(A \cap B) \cup (A \cup B)$

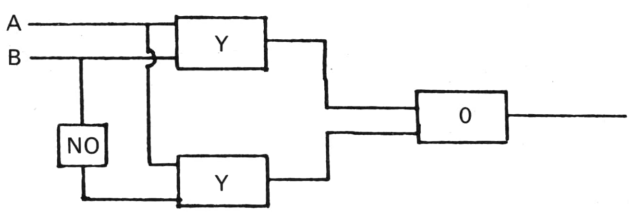
b) $A^c \cap (A \cap B)^c$

c) $(A \cup B)^c \cup (A \cap B)$

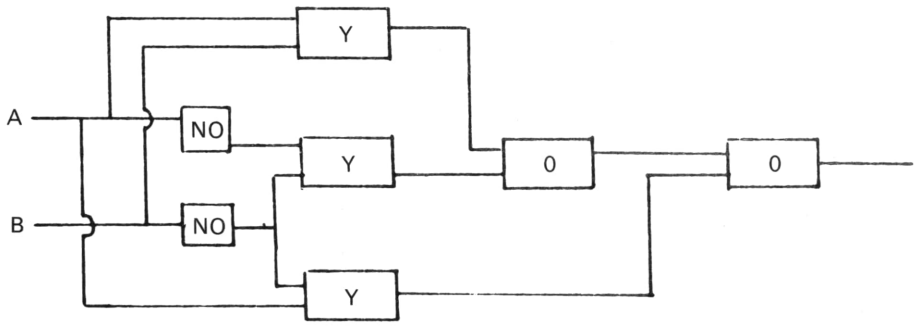
d) $A^c \cap (B \cup A)^c$

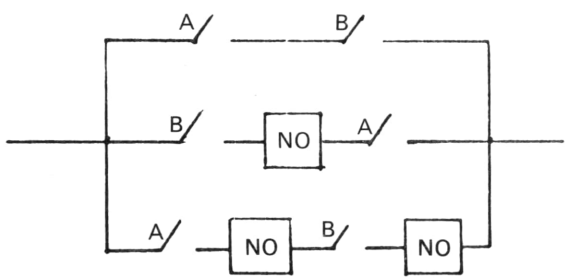
6. Construir circuitos equivalentes a los casos *a, b, c* y *d* del ejercicio 5, a ser posible, más simplificados, aplicando las propiedades de los conjuntos.

7. Construir un circuito equivalente al siguiente, pero más simplificado:



8. Comprobar que los dos circuitos siguientes son equivalentes y obtener una expresión más simplificada:





4

Memoria y control de errores. Soportes de información

4.1. LA MEMORIA

Las máquinas de calcular poseen como unidad de memoria básica unos núcleos de ferrita (o algún biestable electrónico) que recientemente están siendo sustituidos por burbujas magnéticas, que adoptan dos posiciones según estén imantados o no, teniendo la facultad de conservar su estado mientras no lo altera alguna circunstancia externa.

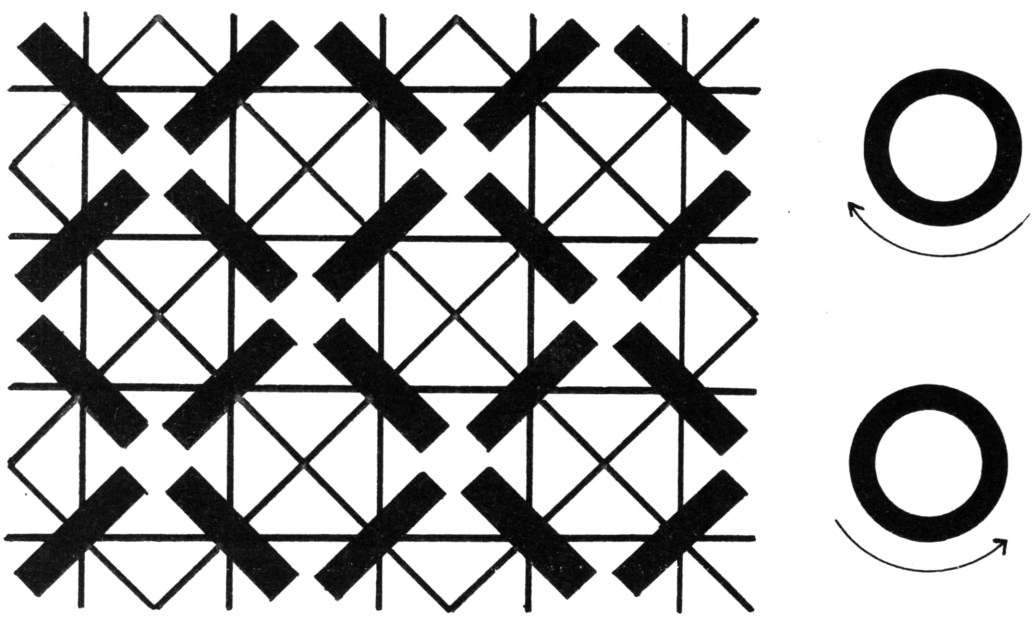


FIG. 14. Fotografía ampliada de núcleos de memoria.

Convenimos en que un núcleo no imantado representará el 0 y el imantado el 1. Estas posiciones elementales reciben el nombre de bit (Binary digit), ya que el 1 y el 0 son los dígitos binarios.

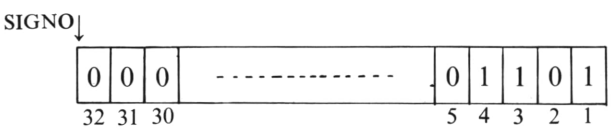
Un conjunto de bits se llama una palabra; ésta suele constar de 8 bits, 16, 32 y hasta 64 bits, aunque es fija para cada máquina. Un conjunto de 8 bits se llama un byte, y es la unidad que suele usarse para establecer la capacidad de memoria de una máquina; así, puede decirse que la capacidad de memoria de una máquina de calcular es 1024 bytes.

Como el byte es una unidad pequeña se suele usar el Kbyte, que es igual a 1024 bytes; así, cuando oímos que un miniordenador tiene 8 Kbytes o sólo 8 K, podemos pensar que dispone, aproximadamente, de unas 8000 memorias de 8 bits cada una.

En algunos modelos, decir sólo 8 K equivale a decir 8 por 1024 palabras, necesitando entonces saber cuál es la longitud de la palabra en dicho modelo.

Supongamos que para manejar números enteros un ordenador usa grupos de 4 bytes, en total 32 bits; el último (o el primero) de estos bits suele reservarse al signo, 0 al + y 1 al −.

La representación del número 13 en una palabra sería:

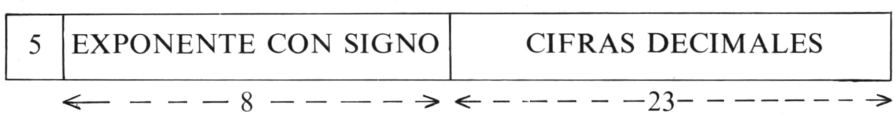


En este caso el mayor número entero que puede almacenarse sería cuando todos los bits son unos; el número binario compuesto de 31 unos, es en decimal:

$$1 + 2 + 2^2 + 2^3 + \cdots + 2^{30} = 2^{31} - 1 = 2147483647$$

es decir, almacena números enteros hasta de diez cifras.

Para los números con punto decimal podría usarse una configuración análoga, pero el punto nos crearía problemas, por lo que la máquina divide los 32 bits en tres partes que pueden ser como sigue:



y trata internamente todos los números decimales en notación exponencial con parte entera cero y la primera cifra decimal distinta de cero (véanse ejemplos 3, 4, 5 y 6), se sitúa en su campo a la izquierda.

Con la configuración anterior, el mayor número almacenable en memoria tendría de exponente siete unos, cuyo valor en base 10 es:

$$1 + 2 + 2^2 + \cdots + 2^6 = 2^7 - 1 = 127$$

y admitiría:

$$1 + 2 + 2^2 + \cdots + 2^{22} = 2^{23} - 1 \simeq \text{siete cifras decimales.}$$

En ocasiones, el ordenador puede aceptar números en representación interna que luego la pantalla no tiene capacidad para mostrarlo (por ejemplo, es evidente que para los resultados con decimales sean correctos y la máquina haga redondeo, interiormente ha de tener registradas más cifras de las que aparecen en pantalla).

Suponiendo que el formato de una palabra es de 4 bytes y la distribución anteriormente estudiada, vamos a hacer una representación interna de algunos números.

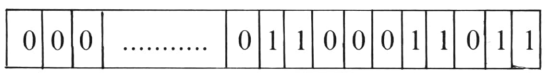
EJEMPLO 1

Representar el número 795

Como

$$795 = 1100011011_{(2}$$

y es positivo se tiene:

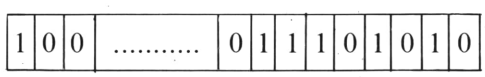


EJEMPLO 2

Representar el número –234.

$$234 = 1101010_{(2}$$

al ser negativo el bit de la izquierda es un 1.



EJEMPLO 3

Representar el número 9.25.

Pasemos 9.25 a base 2

$$9 = 1001_{(2}$$

$$0.25 \cdot 2 = 0.5 \qquad 0$$
$$0.5 \cdot 2 = 1 \qquad 1$$

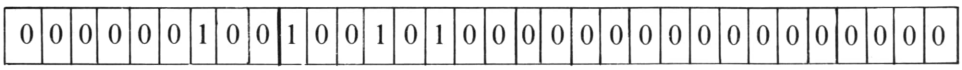
luego:

$$9.25 = 1001.01$$

según dijimos en la representación de números reales la parte entera debe ser cero y la primera cifra decimal distinta de cero; teniendo en cuenta que:

$$1001.01 = 0.100101 \cdot 10^{100} \quad \text{(todo en base 2: } 2 = 10_{(2} \text{ y } 4 = 100_{(2)})$$

la representación sería:



Exp. ⟵ ⟶ CIFRAS DECIMALES

El primer bit de la izquierda es 0, por ser el número positivo; el segundo bit es cero, por ser el exponente positivo; en los siete bits siguientes está el exponente situado a la derecha de su campo y, a continuación, la parte decimal a la izquierda de su campo.

Advertimos que en ocasiones las representaciones internas siguen reglas más sofisticadas, aunque el fundamento sea el anterior, pero se optimiza el rendimiento añadiendo algunas varientes que renunciamos a tratar aquí, por considerar que salen de los objetivos de este libro.

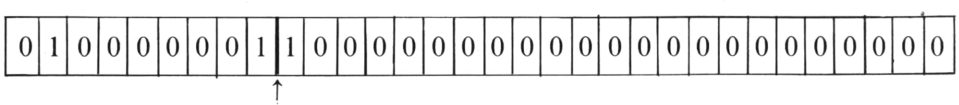
EJEMPLO 4

Representar el número 0.125.

Sabemos que

$$0.125 = 0.01_{(2} = 0.1 \cdot 10^{-1}{}_{(2}$$

la representación es:



EJEMPLO 5

Representar el número –19.4.

Como

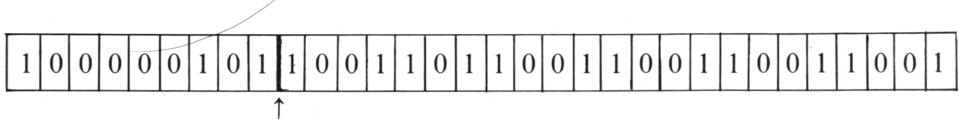
$$19 = 10011$$

y

$$0.4 = 0.\overline{0110}$$

se tiene que:

$$-19.4 = -10011.\overline{0110}_{(2} = -0.100110\overline{0110} \cdot 10^{101}{}_{(2}$$



EJEMPLO 6

Representar el número –0.00625.

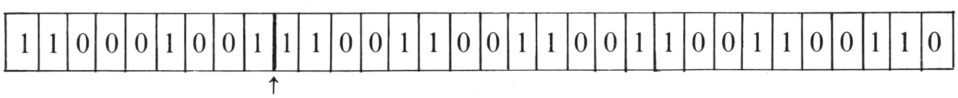
Pasemos este número a base 2:

0.00625 · 2 =	0.01250	0
0.0125 · 2 =	0.0250	0
0.025 · 2 =	0.05	0
0.05 · 2 =	0.1	0
0.1 · 2 =	0.2	0
0.2 · 2 =	0.4	0
0.4 · 2 =	0.8	0
0.8 · 2 =	1.6	1
0.6 · 2 =	1.2	1
0.2 · 2 =	0.4	(se repite)

Así que:

$$-0.00625 = -0.000000\overbrace{0011} = -0.00000001\overbrace{1100} = -0.\overbrace{1100} \cdot 10^{-1001}$$

la representación sería:



4.2. CONTROL DE ERRORES

Los mecanismos electrónicos gozan de bastante seguridad en los movimientos de datos, no obstante, disponen de códigos detectores y en algunos casos correctores de algún posible error.

Pongamos un ejemplo simplista, pero significativo; en lugar de transmitir una información A codificada en n bits, se transmite la misma información A repetida tres veces en $3n$ bits. A la recepción de información pueden presentarse tres eventualidades:

a) Se tienen tres informaciones idénticas: prácticamente con toda seguridad la información es correcta.
b) Dos informaciones idénticas y una diferente: podemos asegurar con mucha probabilidad que hay error en la tercera. Se corrige así el error de la tercera información.
c) Las tres informaciones son distintas: se detecta el error, pero no se corrige.

Otra forma de detectar un posible error en la transmisión de un dato *A* en *n* bits, consistiría en añadir un bit más a esa información llamado bit de paridad, que valga 0 si en la información *A* figura un número par de unos, y 1 si figura un número impar. Este bit de paridad detectaría un error en un solo bit, pero no lo corregiría.

Las correcciones y detecciones reales suelen ser más complicadas de lo aquí expuesto, aunque siguen de alguna forma las ideas anteriores y siempre se juega con probabilidades y con la gran seguridad de los mecanismos electrónicos en la codificación y transmisión de datos.

4.3. SOPORTES DE INFORMACION

El tratamiento de la información supone previamente el almacenamiento de ésta en algún soporte fácilmente accesible, bien para recuperarla o bien para hacer referencia a la misma cuantas veces resulte necesario; podemos, incluso, en algunos casos, alterarla o sustituirla siempre que lo creamos conveniente.

Los soportes de información más usuales son:

— Ficha perforada.
— Cinta de papel.
— Cinta magnética.
— Tambores magnéticos.
— Discos.

a) *Ficha perforada*

La ficha perforada es uno de los medios que han sido usados con más frecuencia para la comunicación con los ordenadores; la información aparece escrita en la parte superior de la ficha y su codificación se registra mediante unos orificios rectangulares situados en lugares predeterminados de la ficha.

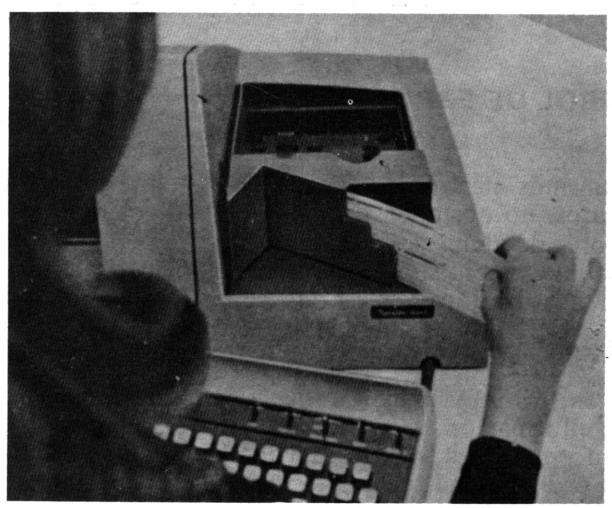


FIG. 15. Colocación de una ficha en la perforadora.

Estos orificios se hacen tecleando una máquina perforadora directamente, aunque también podemos obtener fichas perforadas si pedimos al ordenador que los resultados de algún proceso que esté tratando, los muestre sobre fichas perforadas.

La interpretación de estas fichas por parte del ordenador supone la existencia de una lectora de fichas.

b) *Cinta de papel*

La cinta de papel perforada es análoga a la ficha; los datos se registran en la cinta mediante perforaciones situadas en canales paralelos longitudinalmente, pudiendo variar el número de canales según el modelo de ordenador y de cinta.

La interpretación de los datos contenidos en una cinta perforada debe hacerse mediante una lectora de cintas de papel; asimismo, el registro de datos sobre una cinta de papel necesita de una perforadora para este tipo de cintas.

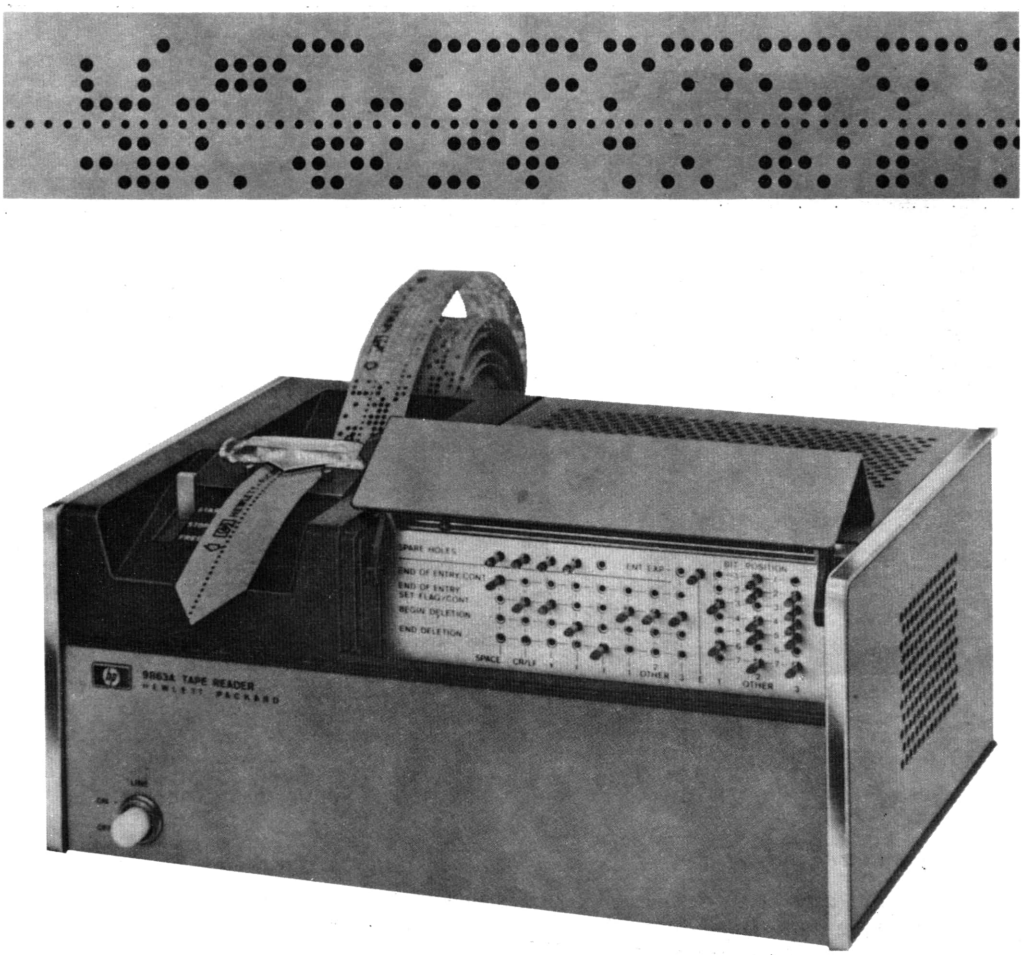


FIG. 16. Cintas perforadas

c) *Cinta magnética*

La cinta magnética se desarrolla a partir de 1952, y consiste en una tira continua de plástico flexible, cuya superficie está recubierta de un material magnetizable. Los datos se registran como puntos magnéticos (bits) en canales o pistas paralelas a lo largo de la cinta; dichos puntos magnéticos tienen la propiedad de conservar su estado y pueden ser interpretados por una lectora de cintas magnéticas.

El ejemplo más simple de cinta magnética es una cinta de un cassette normal que, hoy día, sirve como registro de almacenamiento a muchos microordenadores.

El cassette hace de unidad grabadora y de unidad lectora al ser conectado al ordenador; la grabación en la cinta se hace secuencialmente, lo que exige, a la hora de buscar un dato grabado, que se tenga que buscar a lo largo de la cinta hasta que la cabeza lectora encuentre lo deseado.

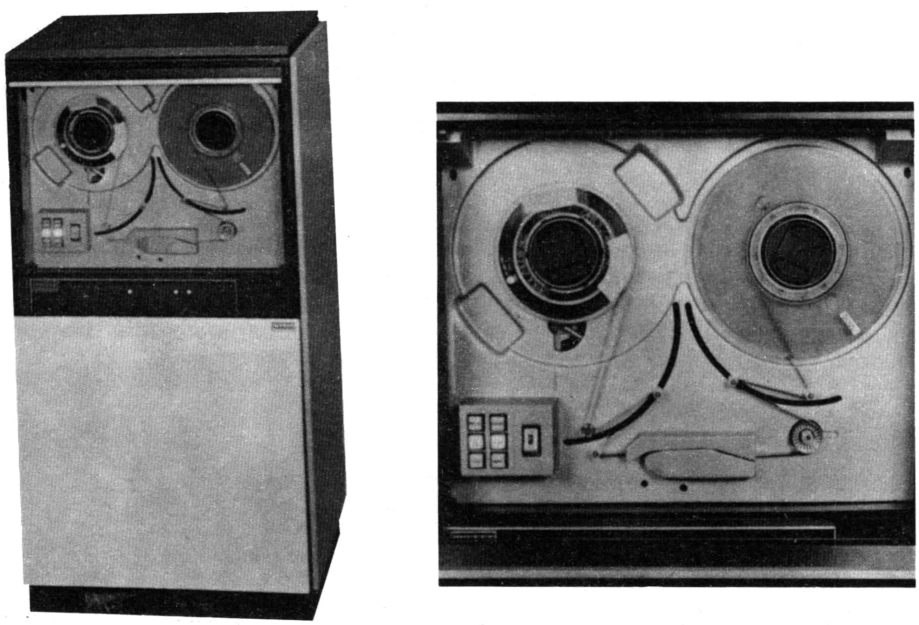


FIG. 17. Unidad de cinta magnética.

d) *Tambores magnéticos*

A igual que la información registrada sobre la cinta, puede hacerse con otro tipo de superficie magnética, como es el caso de los tambores magnéticos; los tambores se desarrollaron a partir de 1952, y constan de un cilindro sobre cuya superficie magnética se graban los datos en pistas paralelas. La

lectura y escritura se consiguen mediante cabezas que leen o graban sobre cada pista del cilindro a medida que éste gira. El sistema de acceso a un dato permite colocar rápidamente una cabeza lectora sobre la pista que se encuentra y una vez sobre ella buscar el dato secuncialmente mientras gira el cilindro.

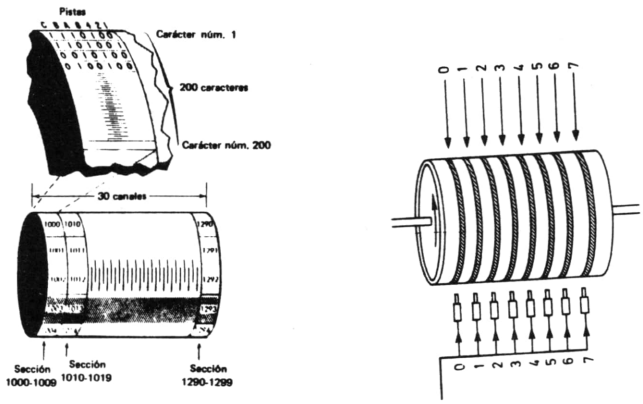


FIG. 18. Tambores magnéticos.

e) *Discos*

Los discos magnéticos son similares a los discos de audición normales, con la particularidad de que los surcos son concéntricos, en lugar de formar una espiral; de esta forma cada surco origina una pista circular. Las caras del disco, a igual que las cintas y tambores, están recubiertas de un material magnetizable. Los datos se graban como puntos magnéticos (bits) a lo largo de cada pista hasta completarla antes de pasar a la siguiente.

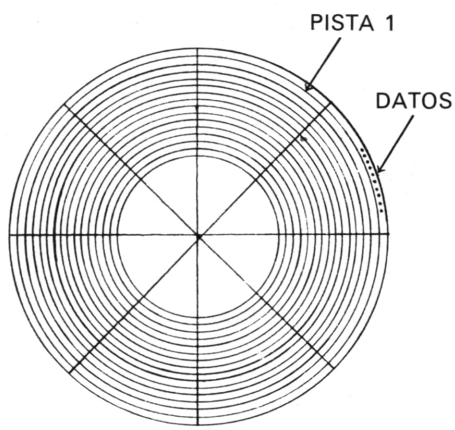


FIG. 19. Disco dividido en 8 sectores.

El número de pistas en cada cara de un disco varía con el modelo; los primeros discos (1956) disponían de 100 pistas circulares, y cada pista tenía capacidad para cinco registros de 100 caracteres cada uno; estos datos han sido en la actualidad ampliamente superados.

La unidad de almacenamiento de discos no suele constar de un único disco, sino de un bloque de ellos.

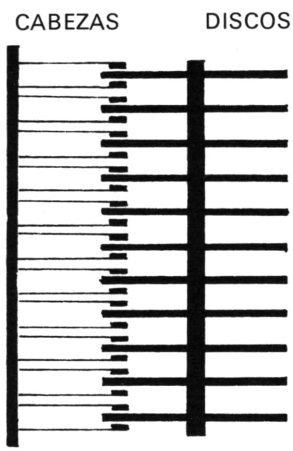


FIG. 20. Cilindro de 11 discos.

La unidad de discos dispone de unas cabezas lectoras (una para cada cara del disco) que pueden ser fijas o móviles y que pueden acceder directamente al lugar que le interese del disco; esta forma de acceso se llama acceso aleatorio que es, lógicamente, mucho más rápido que el acceso secuencial.

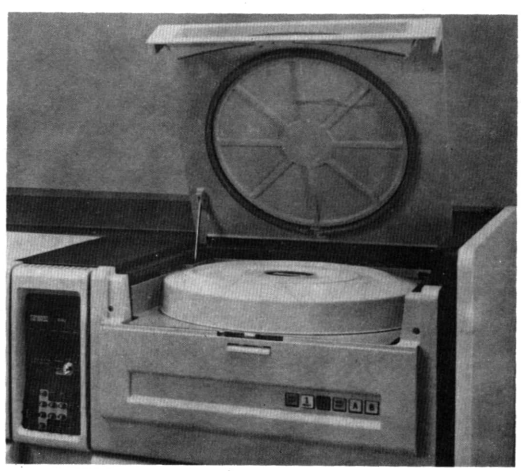


FIG. 21. Unidad de disco.

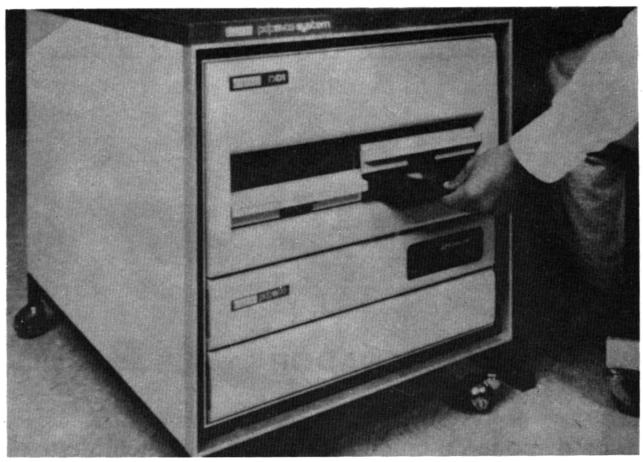


FIG. 22. Unidad de discos pequeños (diskettes).

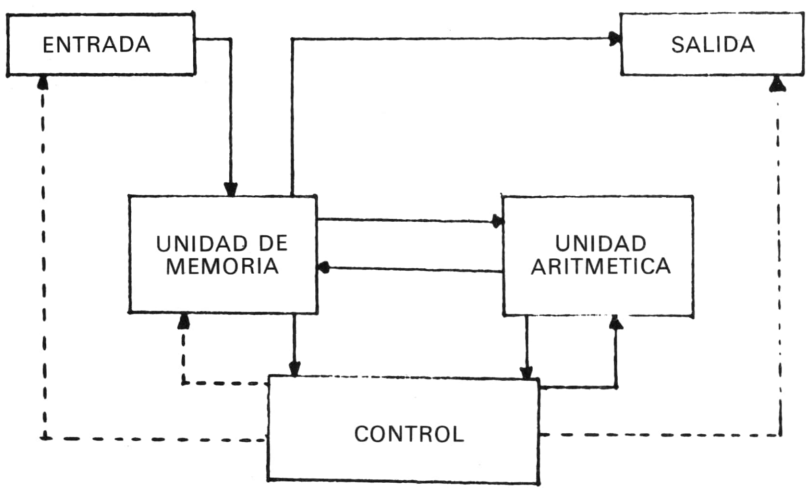
Actualmente, el disco magnético tiende a ser sustituido por otro tipo de disco que, usando rayo láser, evite el mecanismo de contacto de las cabezas lectoras con el disco; esto supone la reducción del tamaño de los surcos o pistas, con un enorme aumento de la densidad de información sobre él y de la rapidez de acceso.

5

Esquema general de un ordenador

5.1. ESQUEMA DE UN ORDENADOR

Todo ordenador dispone de cinco unidades fundamentales:



Unidad de entrada: son los dispositivos que permiten al hombre comunicar datos al ordenador. Según el soporte contenga estos datos, la entrada puede hacerse:

— Directamente tecleando los datos en una consola o terminal.
— Mediante fichas perforadas a través de la lectora de fichas.
— Si los datos están en discos o en cinta habrá que colocar éstos en su lectora correspondiente.
— Usando una lectora óptica si los datos están escritos sobre papel con tinta apropiada.

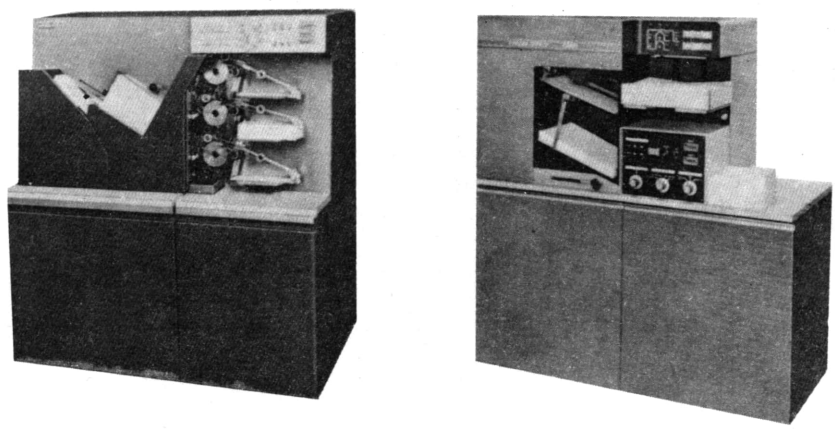


FIG. 23. Lectoras ópticas

Unidad de memoria: es el destino natural de la unidad de entrada; los datos se almacenan internamente donde permanecen fácilmente accesibles. Cada posición de almacenamiento se identifica por un número de posición llamado dirección de almacenamiento; por medio de estas direcciones, el ordenador puede disponer de cualquier dato en el momento que lo necesite y cuantas veces sea necesario.

Unidad aritmética: donde se realizan las operaciones; únicamente pueden ejecutarse las cuatro operaciones elementales y comparaciones entre datos.

Unidad de salida: son los dispositivos que el ordenador puede usar para depositar los resultados de un proceso. Estos resultados pueden:

— Aparecer sobre una pantalla.
— Perforarse en fichas.
— Grabarse en cintas o discos.
— Escribirse en una impresora de papel continuo.

La salida impresa es la más usual cuando se trata de resultados que han de someterse a un estudio o si únicamente se desea tener constancia escrita de ellos.

Desde el año 1949 a 1970 se han desarrollado impresoras de impacto, es decir, impresoras en las que el elemento impresor ha de establecer contacto con el papel para dejar impreso el símbolo deseado. La velocidad de estas impresora ha variado desde un centenar de líneas por minuto en los primeros modelos, hasta unas dos mil líneas por minuto en los últimos.

En 1975, IBM presentó un sistema de impresión que, combinando el rayo láser con la electrofotografía, no funciona con impacto y consigue hasta 13 000 líneas por minuto; hoy día, por este procedimiento, se superan las 20 000 líneas por minuto.

Unidad de control: como su propio nombre indica es la encargada de controlar el trabajo de los demás dispositivos, dar prioridades a los trabajos que le lleguen, pedir datos a memoria, ordenar a la unidad de cálculo que efectúe una operación, ordenar en qué lugar de la memoria puede guardarse el resultado, transferir datos y mensajes a la unidad de salida, pedir datos a la unidad de entrada, etc.

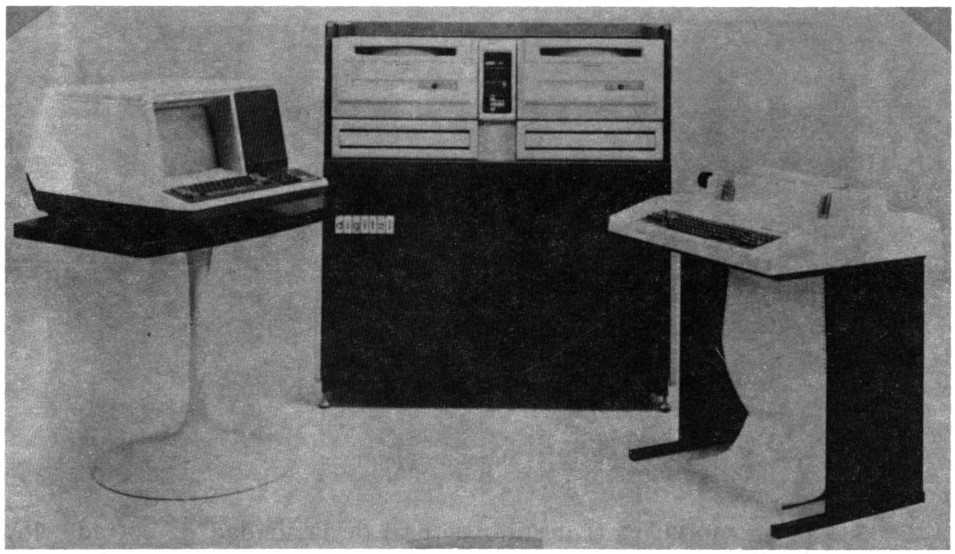


FIG. 24. Equipo compuesto por un terminal con pantalla, dos unidades de disco y una consola con impresora.

5.2. SIMULACION DE UN ORDENADOR EN CLASE

Una experiencia interesante es la simulación de un modelo de ordenador en la clase, donde cada alumno represente una unidad del ordenador y efectuar así, algunas operaciones simples.

Conviene tener en cuenta que:

a) El de entrada sólo puede enviar a memoria lo que control le ordene.
b) El de salida escribirá en la pizarra siguiendo las instrucciones de control.
c) El de memoria tendrá un listín de datos a disposición de control; si la máquina ha de disponer de alguna función como raíz cuadrada, senos o cosenos..., en memoria figurarán las tablas correspondientes.
d) El de la unidad de cálculo sólo efectuará sumas, restas, multiplicaciones, divisiones y comparaciones.
e) Quien haga de control ya sabe que todo, absolutamente todo, ha de pasar por él.

SEGUNDA PARTE

ALGORITMOS Y ORGANIGRAMAS. ESTRUCTURAS BASICAS

6

Algoritmos y organigramas. Lenguajes algorítmicos

6.1. INTRODUCCION

El tratamiento de un problema por ordenador lleva implícito los siguientes puntos:

a) Conocimiento profundo del problema.
b) Contruir un algoritmo que permita llegar a su solución, es decir, diseñar un camino por el que paso a paso nos permita alcanzar los resultados.
c) Opcionalmente, y para clarificar el algoritmo, podemos hacer una representación gráfica con cada uno de sus pasos, que llamaremos organigrama.
d) Escribir el algoritmo en un lenguaje que entienda el ordenador.
e) Pasarlo por el ordenador y recoger los resultados.

Como vemos, desde que se nos plantea el problema hasta que el ordenador nos suministra los resultados hay una serie de pasos que deben hacernos pensar, que el ordenador no es una máquina que acepta y resuelve todo tipo de problemas planteados de cualquier forma.

Lejos de liberar al hombre de un estudio del problema, le obliga a estudiarlo hasta en sus más mínimos detalles (no a ejecutarlo), para que pueda trazar un camino riguroso que llegue hasta su solución (algoritmo) y, si es necesario, hacer un gráfico del proceso. Esta primera fase, apartados *a, b* y *c,* se suele llamar análisis del problema y será para nosotros la más importante, porque al hacerse independientemente del tipo de ordenador que usemos, tiene un carácter de permanencia en el tiempo, del que carecen los apartados *d* y *e*.

No obstante, el comprobar el funcionamiento del algoritmo construido es siempre una garantía de que no nos hemos equivocado, a la vez que un ejercicio de programación en algún lenguaje de ordenador, por lo que también trataremos, a lo largo del libro, la traducción de los algoritmos estudiados a algún lenguaje de uso generalizado.

Así pues, antes de comenzar el estudio de problemas en sus diversas fases, veamos separadamente qué significan cada uno de los siguientes apartados:

a) ¿Qué son los algoritmos?
b) Construcción de organigramas.
c) Lenguajes algorítmicos para ordenadores.

6.2. ALGORITMOS

Un algoritmo es una sucesión de instrucciones para realizar un proceso paso a paso. El tratamiento algorítmico de un problema puede ser opcional en procesos simples, sin embargo, resulta indispensable en cuanto el proceso adquiere cierto grado de complejidad; en cualquier caso, siempre es recomendable su uso. Consideremos algunos ejemplos:

EJEMPLO 1

Supongamos que tenemos una colección de fichas, cada una con un nombre escrito y queremos ordenarlas alfabéticamente; veamos un posible algoritmo que nos resuelve este problema independientemente del número de fichas que tengamos.

Vamos a formar a partir de la colección de fichas dadas, que formarán un bloque inicial A, un nuevo bloque B a donde iremos pasando las fichas de A, de una en una, pero que irán colocándose ordenadamente según van llegando.

Inicialmente, todas las fichas están en A y el bloque B está vacío; tomamos una ficha de A y la colocamos en B; a continuación, tomamos una nueva ficha de A y la situamos en B ordenada con respecto a la que ya estaba; tenemos así dos fichas ordenadas en B; seguidamente, tomamos otra ficha de A y la situamos en B en orden alfabético con respecto a las dos que ya están allí, con lo que tenemos en B tres fichas ordenadas. Así sucesivamente, llegará un momento en que el bloque A quede vacío y en B estarán todas las fichas ordenadas.

Al hecho de situar en B una ficha, en el lugar que le corresponde por orden alfabético lo llamaremos inserción de ficha en el bloque B.

Podemos resumir el proceso, diciendo que consiste en ir tomando fichas de A e insertándolas en B hasta que A quede vacío; no obstante, nos interesa desglosar esta expresión para separar claramente todos los pasos en ella implícitos.

El algoritmo puede expresarse:

a) Tomar una ficha de A.
b) Colocarla en B.
c) Tomar otra ficha de A.
d) Insertarla en B.
e) Si A no está vacío, volver a *c*.
f) Fin.

Los apartados *a* y *b* se refieren al hecho de pasar la primera ficha de A a B, hecho que podemos considerarlo como una inserción en un conjunto vacío, es decir, insertar una ficha en B, cuando en B no hay aún ninguna ficha,

significará sencillamente colocarla en B. Con este acuerdo el algoritmo se reduce a:

a) Tomar una ficha de A.
b) Insertarla en B.
c) Si A no está vacío, volver al apartado *a*.
d) Fin.

De esta forma, el proceso de ordenación de fichas queda descrito claramente, paso a paso, por el algoritmo anterior, sobre el que podemos observar tres características fundamentales de los algoritmos:

1. Todo algoritmo debe ser finito; es decir, debe terminar en un número finito de pasos, ya que de lo contrario no nos solucionaría ningún problema.
2. Cada paso del algoritmo debe estar definido de una forma muy precisa sin que en ningún caso pueda dar lugar a ambigüedades.
3. El algoritmo no soluciona un problema concreto, sino un tipo de problema (en el caso anterior, el de ordenación alfabética).

Conviene hacer notar que en todo algoritmo pueden existir unos parámetros iniciales que llamaremos datos, que habría que valorarlos al aplicarlo a un caso concreto, y unos parámetros finales que hacen referencia a los resutados.

Las tres propiedades anteriores son particularmente importantes cuando se trata de algoritmos que se usan para solucionar problemas por medio de ordenadores, donde la precisión en la forma es fundamental.

Por todo lo dicho anteriormente, queda de manifiesto que la palabra algoritmo no va necesariamente unida a la de ordenadores; esta separación la haremos patente en el ejemplo siguiente.

Ejemplo 2

Supongamos que conocemos el nombre y primer apellido de una persona, así como la dirección del piso en que vive; deseamos saber su número de teléfono, para lo cual disponemos de una guía telefónica; un algoritmo que describe el proceso de búsqueda puede ser el siguiente:

a) Coger la guía telefónica.
b) Buscar sección de teléfonos por calles.
c) Localizar la calle.
d) Localizar dentro de la calle el número del bloque.
e) Localizar dentro del bloque el nombre de esta persona.
f) Anotar el teléfono correspondiente.
g) Fin.

Esta sería la descripción de los pasos más importantes, pero evidentemente podríamos engrosar la lista con otros detalles que ampliarían este algoritmo; esto hace que, especialmente en este tipo de algoritmos no numéricos, para no hacer una lista demasiado grande de pasos necesitamos fijar de antemano un número aproximado de pasos a especificar y seleccionar, con arreglo a ese número, los que a nuestro juicio sean más importantes.

Conviene hacer notar que, tanto en el ejemplo primero como en el segundo, hemos construido «un algoritmo» para el proceso en cuestión, pero dichos algoritmos no son los únicos que resuelven el problema.

EJEMPLO 3

Un algoritmo numérico: se trata de calcular los 25 primeros números impares como términos de la sucesión $a_n = 2n - 1$.

Como sabemos, estos términos se obtienen sustituyendo los sucesivos valores de n: 1, 2, 3, . . . , 25 en la expresión $a_n = 2n - 1$; una primera expresión de este proceso podría ser:

a) hacer n igual a 1.
b) Calcular $a_n = 2n - 1$.
c) Hacer n igual a 2.
d) Calcular $a_n = 2n - 1$.
e) Hacer n igual a 3.
.
.
.

y continuar escribiendo pasos hasta llegar a n igual a 25.

Teniendo en cuenta que n, en cada caso, aumenta una unidad, podemos escribir lo anterior de esta otra forma:

a) Hacer n igual a 1.
b) Calcular $a_n = 2n - 1$.
c) Aumentar n en una unidad.
d) Calcular $a_n = 2n - 1$.
e) Aumentar n en una unidad.
.
.
.

hasta que n tome el valor 25.

Vemos que los apartados *b* y *c* se repiten con idéntica expresion hasta que se calcule a_{25}; la expresión del algoritmo podemos simplificarla a los términos siguientes:

a) Hacer n igual a 1.
b) Calcular $a_n = 2n - 1$.
c) Aumentar n en una unidad.
d) Si n es menor o igual que 25, volver a *b*.
e) Fin.

Obsérvese que si en el paso *d*, sustituimos el número 25 por cualquier otro número K, el mismo algoritmo nos serviría para la obtención de los K primeros números impares. Asimismo, modificando en el paso *b* la expresión de a_n, nos serviría para cualquier otra sucesión.

EJERCICIOS

Escribir algoritmos para los siguientes procesos:

1. Calcular el mayor de un conjunto de números.
2. Preparación de una tortilla francesa de un huevo (8 ó 10 pasos).

3. Sustituir una rueda pinchada de un coche (8 ó 10 pasos).

4. Sumar 50 números sabiendo que únicamente podemos sumar dos números cada vez.

5. Pasar un número natùral en base decimal a base dos.

6. En una llista hemos anotado a lo largo de los cien últimos días cómo nos fue cada día según valoraciones de «Bueno», «Normal» y «Mal». Escribir un algoritmo que nos proporcione:

a) Número de días buenos.
b) Número de días buenos y número de días malos.
c) El mayor número de días normales seguidos.

6.3. ORGANIGRAMAS

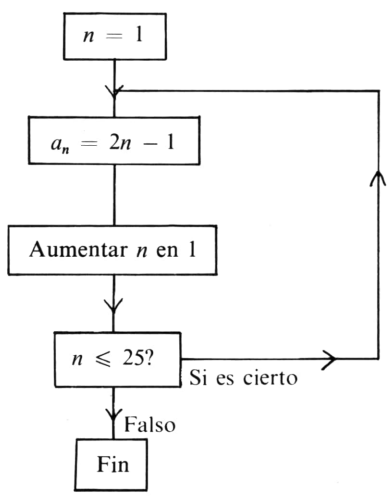
Son representaciones gráficas de los algoritmos que en ocasiones facilitan la comprensión de éste; cada paso del algoritmo se coloca dentro de un bloque y la forma de cada bloque depende del efecto del paso que contenga; además, los bloques están interconectados por flechas que indican la secuencia de los pasos; de ahí los nombres de ordinogramas o diagramas de flujo que se suelen usar también para estas representaciones.

Consideremos el ejemplo 3 del apartado anterior, cuyo algoritmo era el siguiente:

a) Hacer n igual a 1.
b) Calcular $a_n = 2n - 1$.
c) Aumentar n en una unidad.
d) Si n es menor o igual que 25, volver a *b*.
e) Fin.

Un gráfico inmediato a partir de estos pasos es el siguiente:

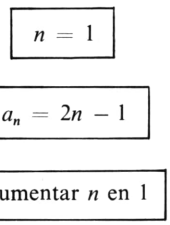
a)
b)
c)
d)
e)



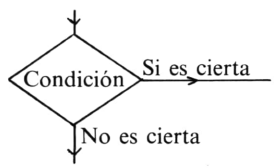
las flechas nos indican en cada caso el camino a seguir.

Sin embargo, los bloques anteriores tienen efectos distintos, y para una visión más clara del proceso convendría dar formas distintas a los bloques según su efecto; es necesario, pues, tomar algunos acuerdos respecto de estas representaciones.

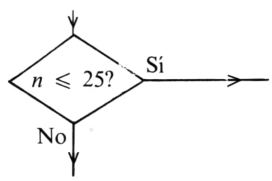
Los bloques cuyo efecto es un cálculo u operación los representaremos por un rectángulo; en nuestro caso serían:



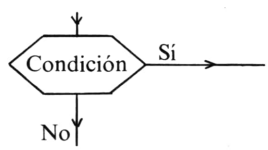
Los bloques de decisión, es decir, aquellos que contienen alguna condición y el camino a seguir varía, según sea cierta o falsa dicha condición, los representaremos por un rombo en forma:



en nuestro caso sería:

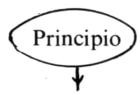


Las salidas, sí y no, pueden invertirse según necesitemos; en ocasiones, si la condición interior al rombo es muy amplia, podemos sustituir el rombo por una figura, tal como:

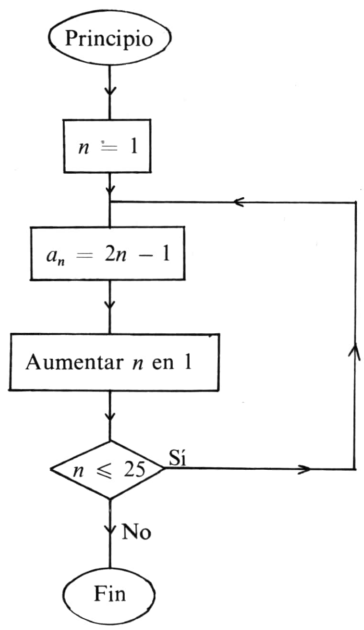


El «Fin» lo colocaremos dentro de una elipse:

además, en todo organigrama, indicaremos un «Principio» de organigrama indicado por otra elipse:



Con estas convenciones, el organigrama anterior quedaría:



En todos los organigramas, y fundamentalmente en los que representan algoritmos para ordenadores, debe figurar en el momento preciso la entrada de datos y salida de resultados.

Continuando con el ejemplo anterior, consideremos que en lugar de obtener los 25 primeros números impares, nos interesa un algoritmo para obtener los K primeros números impares, donde K puede variar en cada caso. El algoritmo es:

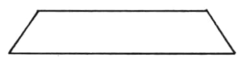
a) Dar un valor a K.
b) hacer n igual a 1.
c) Calcular $a_n = 2n - 1$.
d) Aumentar n en una unidad.
e) Si n es menor o igual que K, volver a c.
f) Fin.

Finalmente, podemos indicar en qué momento debemos anotar un nuevo número impar calculado; esto se haría tras el paso *c* de cálculo y tendríamos la expresión definitiva del algoritmo:

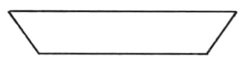
a) Dar un valor a K.
b) Hacer n igual a 1.

c) Calcular $a_n = 2n - 1$.
d) Anotar el valor de a_n.
e) Aumentar n en una unidad.
f) Si n es menor o igual que K, volver a *c*.
g) Fin.

Con respecto al organigrama, todo lo que sea entrada de datos iniciales lo incluiremos en un trapecio de la forma:



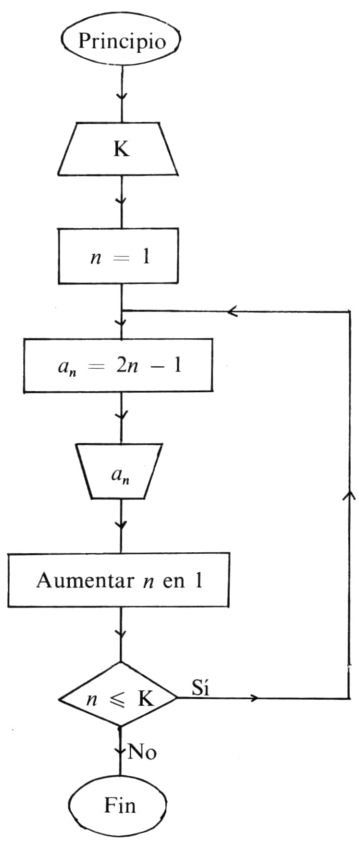
con las variables a definir dentro, y todo lo que se refiere a salida de resultados lo escribiremos en otro trapecio invertido respecto al de entrada:



con las variables de salida en su interior.

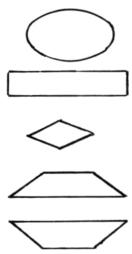
En nuestro caso, el valor a entrar es el de K y los de salida los valores de a_n, con lo que escribiremos:

El organigrama queda finalmente:



En general adoptaremos los siguientes convenios:

1. Dentro de cada bloque incluiremos una única expresión.
2. El seguimiento está indicado por las flechas, que en algunos casos suprimiremos cuando el orden a seguir no dé lugar a confusiones.
3. De cada bloque sale una sola línea, salvo en el bloque de decisión; por otra parte, a cada bloque sólo llegará una línea; si se da el caso de que varios caminos tengan como destino un mismo bloque, éstos deberán formar un nodo previamente del que partirá una única línea hacia el bloque de destino; esto ocurre en el anterior organigrama con el bloque que contiene $a_n = 2n - 1$.
4. Los bloques que usaremos serán de los tipos:



Principio y fin.

Cálculos u operaciones.

Decisiones.

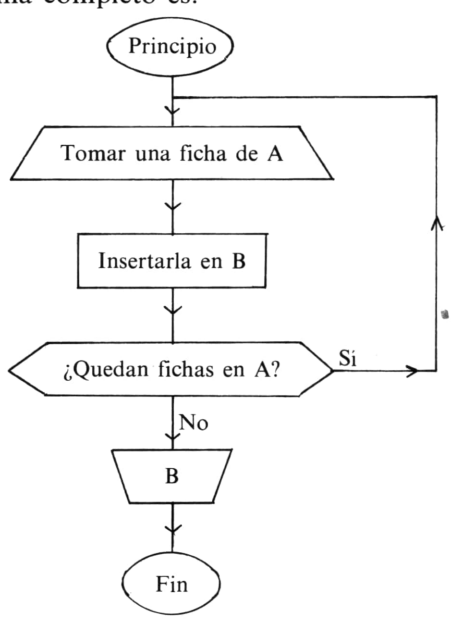
Entrada de datos iniciales.

Resultados.

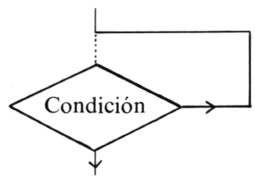
Hagamos el organigrama correspondiente al algoritmo del ejemplo 1 del apartado anterior; dicho algoritmo era:

a) Tomar una ficha de A.
b) Insertarla en B.
c) Si A no está vacío, volver al paso *a*.
d) Fin.

Siguiendo con lo dicho anteriormente, falta un paso de salida de resultados que en este caso sería mostrar el bloque B que contiene todas las fichas ordenadas. El organigrama completo es:



Una construcción que veremos aparecer con mucha frecuencia en los algoritmos y sus correspondientes organigramas es la estructura repetitiva llamada ciclo, que figura en los dos ejemplos anteriores. En el organigrama tiene la forma:



en cada vuelta hay un elemento que se altera y que influye en que la condición sea cierta o falsa.

Estos ciclos constituyen caminos cerrados cuya posible salida debe ser un bloque de decisión; el número de vueltas debe ser fijo (ciclos fijos) o depender de parámetros de entrada u otros valores calculados en el desarrollo del proceso (ciclos variables).

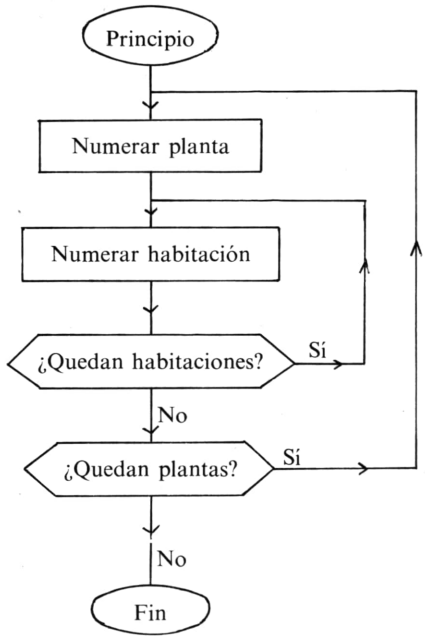
Además, los ciclos pueden encajarse unos dentro de otros tal como ocurre con el siguiente ejemplo: antes de inaugurar un hotel de varias plantas y cierto numero de habitaciones por planta, debemos numerar tanto plantas como habitaciones y lo haremos de la siguiente forma:

a) Numerar una planta que no esté ya numerada.
b) Numerar las habitaciones de esa planta.
c) Si quedan plantas sin numerar, volver al paso *a*.
d) Fin.

De hecho, en el paso *b* hemos incluido varios pasos juntos, a saber, tantos como habitaciones tenga la planta; dividamos *b* en pasos individuales de la forma siguiente:

a) Numerar una planta que no esté ya numerada.
b) Numerar una habitación de la planta donde nos encontramos.
c) Si quedan habitaciones sin numerar en esa planta, volver a *b*.
d) Si quedan plantas sin numerar, volver al paso *a*.
e) Fin.

Este algoritmo contiene dos ciclos encajados, uno comienza en *a* y terminado en *d*, y el otro que comienza en *b* y termina en *c*; en el organigrama se ven con mayor claridad:



EJERCICIOS

Construir los organigramas de los algoritmos propuestos en el apartado 6.1.

6.4. LENGUAJES ALGORITMICOS PARA ORDENADORES

6.4.1. Lenguajes de programación

Dijimos anteriormente que la palabra Informática provenía de INFORmación autoMÁTICA, es decir, tratamiento automático de la información. En un principio, la informática no fue más que unos avances tecnológicos que dieron lugar a unas máquinas, tales que con su uso el hombre se descargaba de tareas fáciles, pero normalmente largas y pesadas; aparece, por tanto, como herramienta de trabajo.

Actualmente sigue teniendo esa función, pero el campo de aplicaciones que cubre es tan amplio y la complejidad de los ordenadores que en la actualidad se fabrican tan grande que, de por sí, podemos considerarla como una ciencia en continuo avance y con dos líneas fundamentales de investigación: investigación en hardware e investigación en software.

Veamos el significado de estos dos conceptos sin traducción literal exacta.

Se llama hardware a todo el conjunto de elementos físicos que constituyen el ordenador: circuitos, interruptores, semiconductores, etc.; la investigación en hardware es, por tanto, la que permite el avance tecnológico en la construcción de ordenadores.

Cuando hablamos de software, nos estamos refiriendo al conjunto de programas que, elaborados por el hombre y una vez registrados en la memoria del ordenador, permite ejecutar diversidad de trabajos. La investigación en software es, pues, el estudio, programación y organización de programas.

Todo ordenador dispone de un software básico que está íntimamente relacionado con su hardware y que permite la explotación de éste; el software básico recibe el nombre de sistema operativo. Un conocimiento a fondo del sistema operativo del ordenador es fundamental para poder explotar en profundidad sus recursos.

Programar un ordenador es escribir un conjunto de instrucciones u órdenes, de tal forma que una vez que el ordenador las ejecute produzca los resultados deseados. La programación de las primeras máquinas se efectuaba a base de un fuerte conocimiento de la estructura de las mismas y de la función de cada uno de sus órganos. Los lenguajes de programación, entonces, eran lenguajes orientados a la máquina.

Así, por ejemplo, para sumar dos números, teníamos que dirigir el almacenamiento de éstos a determinadas posiciones de memoria y después dar el código de la suma y las direcciones de memoria donde los sumandos estaban registrados; volver a dar una nueva dirección de memoria donde el resultado debía ser almacenado.

Afortunadamente, todos estos pasos y algunos más han sido incorporados al software básico del ordenador, lo que hace la labor de programar menos laboriosa. Actualmente, algunas máquinas programables, y opcionalmente los ordenadores, se programan en un lenguaje próximo al lenguaje máquina; esta forma de programar es, como acabamos de decir, altamente incómoda; de aquí que aparezcan lenguajes simbólicos, que permiten escribir las instrucciones del programa en un lenguaje próximo al lenguaje natural.

El nivel de un lenguaje es la proximidad de su expresión al lenguaje natural; así, lenguajes de bajo nivel son lenguajes próximos al lenguaje máquina y lenguajes de alto nivel, los próximos al lenguaje natural.

Existe una gran variedad de lenguajes simbólicos, que provienen a su vez de las distintas aplicaciones a las que van dirigidos.

Así, para aplicaciones científicas tenemos el PL/1, Algol, Basic, Fortran, Pascal, etc.; para aplicaciones de gestión, el Cobol, Rpg, PL/1, etc.; existen lenguajes orientados a la investigación lingüística como el Snobol, lenguajes de simulación como el Mimic, etc.

Un ordenador puede usar varios lenguajes con tal de que tenga incorporado el traductor (ensamblador, compilador o generador) correspondiente a dicho lenguaje; este traductor, transcribe nuestras órdenes al lenguaje máquina, que es el único con el que el ordenador opera.

De los lenguajes algorítmicos para ordenadores, el Basic, como su propia palabra indica, «Beginner's All-purpose Symbolic Instruction Code», es decir, «código de instrucciones simbólicas de carácter general para principiantes», es uno de los lenguajes más simples de alto nivel presente en la mayoría de los ordenadores y que a nivel de enseñanza media es usado en un elevado tanto por ciento de los casos, por su gran facilidad y versatilidad. Puede enseñarse

sin estar limitado a un ordenador concreto, como veremos, adaptándose luego al usuario, a cualquier ordenador con suma facilidad.

A pesar de su simpleza, veremos cómo se podrá acceder y ejecutar algoritmos de un elevado grado de complejidad, así como con mucha sencillez iniciarse en otros lenguajes que necesite en el futuro, dado que siempre existen determinadas estructuras mentales que se repiten en todos ellos.

Será el lenguaje Basic el que en sus aspectos más generales usemos para la programación de los algoritmos planteados. Tanto para lo que se refiere a este lenguaje como para la simplificación de las expresiones de los algoritmos y organigramas (sobre todo en las fórmulas) vamos a establecer unas reglas mínimas sobre los operadores y funciones que consideraremos en nuestras expresiones.

6.4.2. Operadores aritméticos

Los operadores usados en Basic son los siguientes:

\+ para la suma;
\− para la resta;
* para el producto;
/ para la división;
↑ para la potenciación.

En las expresiones que usemos debemos indicar siempre el tipo de operador que se utiliza; así, por ejemplo, evitaremos el producto implícito y escribiremos 2*A en lugar de 2A.

6.4.3. Expresiones aritméticas y funciones

El concepto de variable en Basic es análogo a su uso matemático; en nuestro caso una variable va a representar una memoria cuyo contenido es susceptible de variación. Así, la variable C va a representar una posición en memoria cuyo contenido puede variar, pero al que siempre nos referiremos por C. Las variables que usaremos, en principio, serán las letras de la A a la Z, excepto la Ñ.

Las constantes serán: Ø, 1, 2, 3, 4, 5, 6, 7, 8 y 9 y todas las que se puedan expresar con estos dígitos, es decir, números enteros, decimales (usando el punto decimal), exponenciales (usando la E de exponencial). Usaremos también la constante π.

Ejemplos de constantes son:

3452	ØEØ (=Ø)
23.75	15.7564756
32E5 (=$32 \cdot 10^5$)	−1743271E25 (=$-1743271 \cdot 10^{25}$)
Ø	4.37E−24 (=$4.37 \cdot 10^{-24}$)

El máximo número de posiciones que podrá ocupar una constante será de once, incluido el signo.

Los paréntesis y otros signos de separación y puntuación los escribiremos de la forma usual:

( ) . , ; : ”

Las funciones que usaremos y que son generales a todos los modelos de ordenadores son:

SIN(X)	seno de X en radianes.
COS(X)	coseno de X en radianes.
TAN(X)	tangente de X en radianes.
ASN(Y)	arcoseno de Y.
ACS(Y)	arcocoseno de Y.
ATN(Y)	arcotangente de Y.
LN(X)	logaritmo neperiano de X.
LOG(X)	logaritmo decimal de X.
EXP(X)	exponencial de base e; calcula e^x.
INT(X)	parte entera de X, es decir, el mayor número entero que sea menor o igual que X.
ABS(X)	valor absoluto de X.
SGN(X)	función signo de X; su valor es: 1 si X es positivo; Ø si X vale cero; −1 si X es negativo.
SQR(X)	calcula la raíz cuadrada de X.

Todas estas funciones se aplican a una constante, a una variable o a una expresión aritmética del tipo que veremos más adelante.

EJEMPLOS:

Sea A $\doteq$ 2, B = −3 y C = Ø, se tiene:

LN(A) = Ø.693114
SGN(A−B) = 1
SGN(B) = −1
SQR(25+3*B) = 4
EXP(1) = 2.718281 (=e), no confundir con E1 = 10^1
INT(22.37) = 22
INT(−3.5) = −4
SIN(A) = Ø.090929742
ABS(B) = 3
LOG(1ØØ) = 2

Una expresión aritmética, es un conjunto de variables, constantes y funciones aritméticas ligadas por los operadores aritméticos. Son expresiones aritméticas:

2*A−B+4.35*C
SIN(C−4*A)
5*COS(6*A−1)
7*8/9−LN(X)
(A+B)/(X−Z)

Llamaremos instrucción o fórmula aritmética a una variable igualada a una expresión aritmética:

$$v = e$$

v representa una variable y *e* una expresión aritmética; la variable siempre ha de estar a la izquierda y la expresión a la derecha.

EJEMPLOS:

```
B=2*H+5/L
V=9/5
M=9*COS(6*J)-INT(A-18)
Y=T-2
Z=LOG(34.67)-N/(P-Q)
K=4*R↑3
```

Al ejecutarse una de estas instrucciones, el ordenador evalúa la expresión aritmética de la derecha y el resultado se lo asigna a la variable de la izquierda; por este motivo, las variables que aparezcan en la expresión aritmética tienen que haber sido definidas previamente, porque, en caso contrario, el ordenador no podría valorar la expresión o lo haría erróneamente, con resultados imprevisibles para el programa.

Tomando al pie de la letra lo que acabamos de decir vemos que tienen un sentido exacto las instrucciones tales como:

```
T=T-1
B=A*B-B
```

En la primera de estas dos instrucciones, al valor de T le resta 1 y el resultado lo almacena nuevamente en T, siendo «machacado» el que tuviese anteriormente. Tenemos entonces que una variable conserva el valor que se le haya asignado en último lugar.

A la hora de evaluar una expresión aritmética conviene tener en cuenta la prioridad de los distintos operadores, la cual viene dada por:

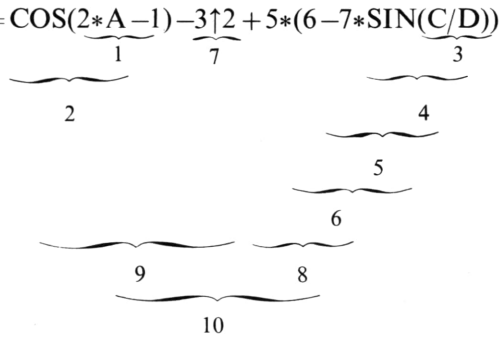
a) Llamada a funciones aritméticas.
b) Signos + y − precedidos de ↑.
c) Exponenciales.
d) Signos + y − unitarios ($-3\uparrow 2 = -(3^2)$).
e) * y / .
f) + y −.

En caso de igual prioridad, el orden de ejecución de las operaciones es de izquierda a derecha; esto último puede modificarse con el uso de paréntesis, que tienen prioridad sobre el resto y cuyo uso no debe ser escatimado en caso de duda; si hay varios paréntesis, la preferencia corresponde a los más interiores.

EJEMPLOS:

```
K=A*B/C     se efectúa de izquierda a derecha, equivale a (A*B)/C
P=A/B*C     se efectúa de izquierda a derecha, equivale a (A/B)*C
P=A/(B*C)   calcula A/(BC)
```

Vemos cuál sería el orden de ejecución de las operaciones en la siguiente expresión:



El paréntesis que se comienza a evaluar debe terminarse antes de salir de él.

Podemos comprender ahora, que ante una expresión tal como A=B−C, la unidad de cálculo o unidad aritmética la ejecuta, pero la unidad de control es la que dirige y facilita a la unidad de cálculo:

a) Código de operación: la resta en este caso.
b) Dirección en memoria del primer operando: B.
c) Dirección en memoria del segundo operando: C.
d) Dirección donde se debe guardar el resultado A

Afortunadamente en nada de esto intervenimos nosotros.

EJERCICIOS

1. De las siguientes instrucciones en Basic, indicar las que son correctas:

C=B−A	S=A(B−C)	S=T=Q−2
P=17.5*L−7	T=SQR(C↑2−1)	H=P*[4−5*(6−C)]
R=4,657	2*K=P−7	COS(A+T)=Z
V=V*(V−2)	Q=43.7*A−2↑5	L=L−78

3. Los siguientes números están expresados incorrectamente en Basic; explicar por qué:

0,00018	36.47−E12	−0.437E−.6	18'427
7.434.532	0.0000000000019	0.17^{24}	6.432'43

3. Escribir en Basic los siguientes números:

34000000000000000	$-5 \cdot 10^{-18}$	6'437
$129 \cdot 10^{14}$	0.000000000000000076	18.421'56

3. Pasar a notación matemática las siguientes instrucciones:

```
R=(B↑2−K)/(7*A+5)
T=((3*C+5*A)↑4−Y)↑(1/3)
Q=(18*P−45)/K
H=ABS((34−A)/T)
J=SIN(P−Q)/(S↑3)
```

5. Escribir en Basic las expresiones siguientes:

$$\frac{17\left(\frac{3}{4}\right)^2 - \frac{4}{5}}{\frac{1}{2} - 7} \qquad \frac{\left(\frac{4}{3}\right)\sqrt{7} + 18 \cdot \frac{13}{5}}{\left(2 - \frac{3}{4}\right)^{1/5}} \qquad \frac{\left(\frac{1}{13}\right)^3 \cdot \left(\frac{4}{3}\right)^2 - \frac{2}{5}}{\sqrt{\left(7 + \frac{3}{5}\right)^3}}$$

6. Escribir el orden de ejecución de operaciones en las expresiones siguientes:

(A−2*B)↑5−7+6*INT(4−C)
SQR(C↑2−3*L)+5
(4*A−B)↑(C↑2−5)+8

7. Escribir en Basic las siguientes fórmulas:

$e = e_0 + vt$ ecuación del movimiento rectilíneo uniforme.

$e = e_0 + v_0 t + \frac{1}{2} at^2$ ecuación del movimiento uniformemente variado.

$v = \frac{at}{\sqrt{1 + \left(\frac{at}{c}\right)^2}}$ velocidad en un movimiento acelerado para la teoría de la relatividad.

$y = x \tan (p) - \frac{1}{2} g \frac{x^2}{v_0^2 \cos^2 p}$ ecuación de la trayectoria de un proyectil en el vacío.

$e = \frac{1}{2} mv^2$ eneergía cinética.

$\mathrm{T} = \frac{1}{2\pi} \sqrt{\frac{\mathrm{K}}{m}}$ periodo del movimiento armónico simple.

$f = \frac{e^2}{4\pi \varepsilon r^2}$ fuerza de atracción culombiana.

8. Expresar en Basic las siguientes fórmulas:

$$z = \left(\frac{2b - a - 5c}{3x}\right)^4 \qquad t = a - \frac{b}{c} + d$$

$$v = \left(\frac{2b - a - 5c}{3x}\right)^5 \cdot \frac{b}{c} \qquad g = \log \frac{a - b}{c}$$

$$c = \frac{a + b \cos b}{\cos a - 2b} \qquad g = \frac{\ln a - b}{c}$$

$$j = \frac{a - b}{c + d} \qquad g = \frac{\ln(a - b)}{c}$$

$$t = a - \frac{b}{c + d} \qquad d = (2a - b)^{\frac{4k - 2}{3}}$$

$$t = \frac{a - b}{c} + d \qquad e = \frac{\text{arctg } x}{x^2 + 1} - 2 \text{ sen } t$$

7

Instrucciones algorítmicas básicas

7.1. INTRODUCCION

Todos los problemas que sean estudiados de ahora en adelante, se tratarán en los tres aspectos sobre los que insistimos en el capítulo anterior, unidos a la fase inicial de planteamiento del problema, es decir:

a) Planteamiento del problema.
b) Elección del algoritmo a seguir.
c) Constitución del organigrama (en la mayoría de los casos).
d) Programación del algoritmo en el lenguaje Basic.

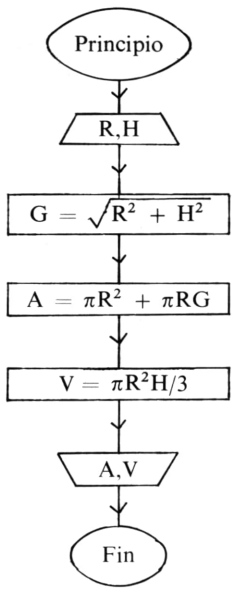
7.2. ALGUNOS PROCESOS SIMPLES SIN RUPTURA DE SECUENCIA

Imaginemos que alguien nos pide que le resolvamos el problema de calcular el área total y el volumen de un cono conociendo el radio de la base R y su altura H.

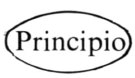
El algoritmo a seguir sería el siguinte:

a) Preguntamos el valor de R y de H.
b) Calculamos la generatriz del cono: $G = \sqrt{R^2 + H^2}$.
c) Calculamos el área: $A = \pi R^2 + \pi RG$.
d) Calculamos el volumen: $V = 1/3\ \pi\ R^2 H$.
e) Mostramos los resultados de A y V.
f) Fin.

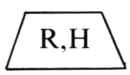
El organigrama sería:



Hecho el análisis del problema, vamos a comunicárselo a un ordenador en Basic:

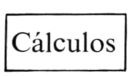


Esto solamente lo usaremos en el organigrama; en el programa no es necesario indicar el comienzo, ya que éste es implícito a la primera instrucción escrita.



Para la entrada de datos usaremos la palabra INPUT; en nuestro caso escribiremos simplemente:

```
INPUT R,H
```

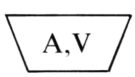


Los cálculos los escribiremos tal cual están añadiendo los operadores que faltan:

```
G=SQR(R↑2+H↑2)
```

o bien, poniendo LET delate:

```
LET G=SQR(R↑2+H↑2)
```



Para la salida de datos usaremos la plabra PRINT, es decir:

```
PRINT A,V
```

El fin lo indicaremos en el programa por END.

Las intrucciones conjuntas de este proceso serían:

```
INPUT R,H
G=SQR(R↑2+H↑2)
A=π*R↑2+π*R*G
V=π*R↑2*H/3
PRINT A,V
END
```

Por cuestión de orden, para nosotros y para el ordenador, vamos a poner delante de cada instrucción un número llamado etiqueta de la instrucción. Los números de etiqueta han de ser mayores que uno e ir creciendo con las instrucciones, aunque no es necesario que sean consecutivos; es más, no es conveniente que sean consecutivos, para, como veremos más adelante, poder intercalar alguna instrucción olvidada.

Los números de etiquetas que usaremos en el texto estarán comprendidos entre 1 y 999, aunque el límite superior varía con el modelo de ordenador.

Así, en nuestro caso podemos escribir:

```
1Ø INPUT R,H
2Ø G=SQR(R↑2+H↑2)
3Ø A=π*R↑2+π*R*G
4Ø V=π*R↑2*H/3
5Ø PRINT A,V
6Ø END
```

Estas seis instrucciones constituyen un programa en Basic; a continuación, las escribimos sobre un teclado de ordenador y el programa queda registrado en la memoria de éste.

Para ejecutar el programa escribiremos en el teclado RUN 1Ø y nos pedirá los valores de R y H, mostrándonos a continuación los valores correspondientes de A y V. Como el programa queda almacenado en memoria, podemos repetir este proceso tantas veces como queramos.

El programa anterior es el mismo para cualquier ordenador que use Basic; únicamente cuestiones puramente externas como: variación de la disposición del teclado, símbolos de operaciones, etc., que son propios para cada modelo, conllevan muy ligeras diferencias entre la ejecución del programa anterior en uno u otro ordenador.

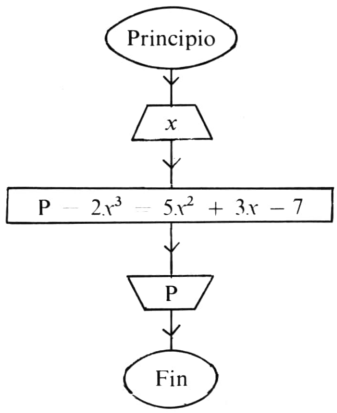
Veamos algunos ejemplos:

1. Hallar el valor numérico del polinomio $P(x) = 2x^3 - 5x^2 + 3x - 7$ en un punto x.

El algoritmo es evidente:

a) Dar un valor a x.
b) Calcular el valor de P en x.
c) Imprimir el valor de P obtenido.
d) Fin.

El organigrama y programa son:




```
1ØØ INPUT X
11Ø P=2*X↑3+5*X↑2+3*X-7
12Ø PRINT P
13Ø END
```

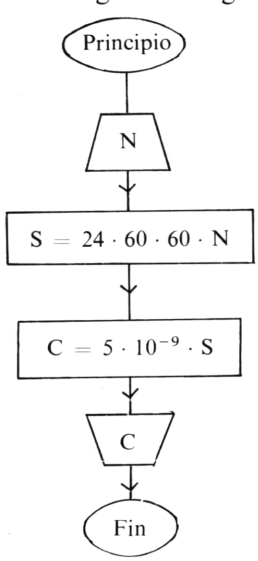
Si deseamos correr este programa y el anterior no lo hemos borrado tendremos que llamarle escribiendo RUN 1ØØ, con lo que indicamos al ordenador que ejecute todas las instrucciones desde la número 1ØØ hasta que se encuentre con un END.

2. La velocidad de crecimiento del cabello humano se estima en $5 \cdot 10$ m/seg. Escribir el algoritmo y el programa correspondiente, de forma que dando los días transcurridos desde el último corte de pelo nos muestre el crecimiento total durante ese periodo.

El algoritmo sería:

a) Entrar el número de días transcurridos.
b) Hallar el número de segundos que suponen estos días.
c) Multiplicar los segundos por $5 \cdot 10^{-9}$.
d) Mostrar el último resultado obtenido.
e) Fin.

Si designamos por N el número de días y por S los segundos equivalentes, tendremos el siguiente organigrama y programa:




```
6Ø INPUT N
7Ø S=24*60*60*N
8Ø C=S*5E-9
9Ø PRINT C
1ØØ END
```

No puede haber dos instrucciones con el mismo número de etiqueta, pues la que escribamos en último lugar machaca a la anterior, es decir, si escribimos varias instrucciones con el mismo número de etiqueta sólo permanece en memoria la última; este hecho suele usarse para corregir instrucciones.

Para borrar la memoria de programas introduciremos la palabra NEW.

3. Un movimiento rectilíneo responde a la ecuación $e = 3t^2 + 5t + 1$, donde e representa el espacio recorrido en un tiempo t; queremos obtener la velocidad media entre dos momentos dados t_1 y t_2.

La velocidad media es el cociente:

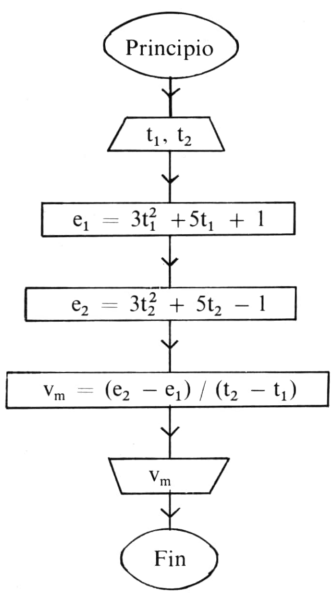
$$v_m = \frac{e_2 - e_1}{t_2 - t_1}$$

siendo e_1 el espacio recorrido hasta el momento t_1 y e_2 el espacio hasta el momento t_2.

El algoritmo sería:

a) Dar valores a t_1 y t_2.
b) Hallar el espacio recorrido en el tiempo t_1.
c) Hallar el espacio recorrido en el tiempo t_2.
d) Calcular la velocidad media.
e) Mostrar v_m.
f) Fin.

Para el programa llamaremos A a t_1, B a t_2, C a e_1 y D a e_2; el organigrama y programa quedarían:




```
4Ø INPUT A,B
45 C=3*A↑2+5*A+1
5Ø D=3*B↑2+5*B+1
55 V=(D-C)/(B-A)
6Ø PRINT V
7Ø END
```

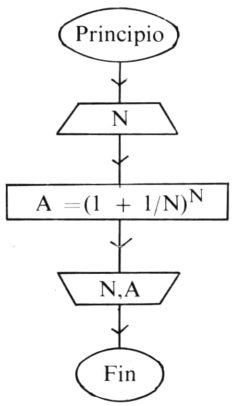
4. Calcular el término N de la sucesión que define el número e.

$$a_n = \left(1 + \frac{1}{n}\right)^n$$

El algoritmo es inmediato:

a) Dar un valor a N.
b) Calcular el valor A del término a_N.
c) Mostrar el valor de N y de A.
d) Fin.

Organigrama



Programa

```
 4 INPUT N
 8 A=(1+1/N)↑N
13 PRINT N,A
18 END
```

En salida nos muestra el valor de N que fue introducido y el valor del término a_N que fue asignado a A. Cada vez que llamemos al programa podemos calcular el término que nos interese.

7.3. SALTOS INCONDICIONALES

Volvamos a considerar el ejemplo 4 del apartado anterior, que nos daba el término N dela sucesión $a_n = (1 + 1/n)^n$. Cada vez que deseemos obtener un término de la sucesión, tendremos que llamar al programa, lo cual resulta bastante incómodo, sobre todo si nos interesa «observar» la evolución de esa sucesión término a término.

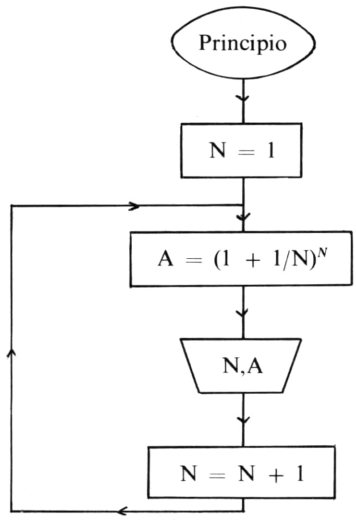
El algoritmo para poder conseguir esta observación sería:

a) Hacer N igual a 1.
b) Calcular el valor de A del término a_n.
c) Imprimir los valores de N y A.
d) Aumentar N en una unidad.
e) Volver a *b*.

Realmente, así planteado, esto no es un algoritmo, porque no sería un proceso finito; no obstante, como se trata de «observar la evolución de los términos de una sucesión», admitiremos momentáneamente este tipo de algoritmo y supondremos que al aplicarlo a un ordenador, éste dispone de una tecla de STOP que permite parar manualmente el proceso cuando nos interese. En el apartado siguiente veremos la forma de evitar esto.

El paso *e* del algoritmo es llamado un salto incondicional, y en el

organigrama se representa uniendo el punto de inicio del salto con el punto de llegada mediante una flecha. El organigrama del anterior algoritmo es:



En el programa, este salto es una instrucción de forma:

ne GO TO m

donde *ne* es el número de etiqueta de esta instrucción, donde se inicia el salto,m y *n* es el número de etiqueta de la instrucción de llegada.

El programa es, por lo tanto:

```
1Ø N=1
2Ø A=(1+1/N)↑N
3Ø PRINT N,A
4Ø N=N+1
5Ø GO TO 2Ø
```

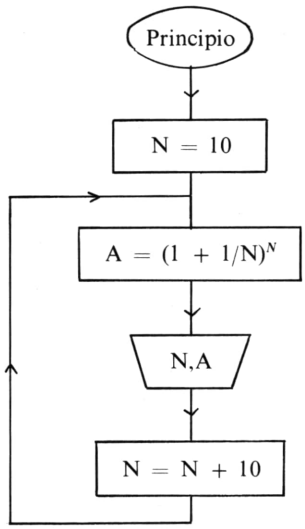
Aparecerían, consecutivamente, todos los términos de la sucesión; el final no se alcanzaría en el programa que sólo puede ser interrumpido manualmente, como se dijo antes.

Continuando con la sucesión a_n anterior, pudiera ocurrir que no nos interese obtener uno a uno todos sus términos, sino que, pongamos por caso, nos interesa que nos muestre los términos de la sucesión de diez en diez, es decir, los términos a_{10}, a_{20}, a_{30}, ...

Todo se reduce a dar a N el valor inicial 10, y después indrementar este valor de 10 en 10 unidades.

El algoritmo, organigrama y programa serían:

a) Hacer N igual a 10.
b) Calcular el valor de A del término a_n.
c) Imprimir los valores de N y A.
d) Aumentar N en 10 unidades.
e) Volver a *b*.

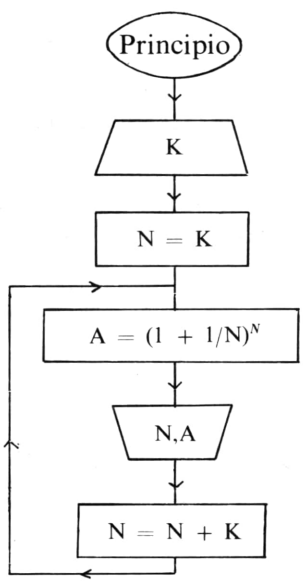



```
1Ø N=10
2Ø A=(1+1/N)↑N
3Ø PRINT N,A
4Ø N=N+10
5Ø GO TO 2Ø
```

Demos un paso más; a raíz del proceso anterior es fácil imaginar cómo sería un proceso que nos diese los términos de la sucesión de k en k, es decir, a_k, a_{2k}, a_{3k}, ... pudiendo variar k en cada ejecución. Lógicamente, k ha de figurar como un elemento a entrar y debe sustituir al número 10, donde aparecía éste en el proceso anterior.

Teniendo en cuenta lo anterior, el algoritmo, programa y organigrama son:

a) Dar un valor a K.
b) Hacer N igual a K.
c) Calcular el valor A del término a_N.
d) Imprimir los valores de N y A.
e) Aumentar N en K unidades.
f) Volver a *c*.




```
1Ø INPUT K
2Ø N=K
3Ø A=(1+1/N)↑N
4Ø PRINT N,A
5Ø N=N+K
6Ø GO TO 3Ø
```

Finalmente, y como última etapa, queremos obtener los términos de la sucesión de k en k, pero a partir de un cierto término m de la sucesión, es decir:

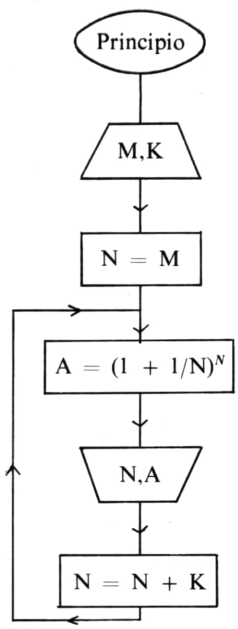
$$a_m,\ a_{m+k},\ a_{m+2k},\ a_{m+3k},\ \ldots$$

en este caso, tanto m como k deben figurar como elementos de entrada.

El algoritmo es:

a) Dar un valor a M y a K.
b) Hacer N igual a M.
c) Calcular el valor A del término a_N.
d) Imprimir los valores de N y de A.
e) Aumentar N en K unidades.
f) Volver a *c*.

Organigrama



Programa

```
1Ø INPUT M,K
2Ø N=M
3Ø A=(1+1/N)↑N
4Ø PRINT N,A
5Ø N=N+K
6Ø GO TO 3Ø
```

7.4. DECISIONES

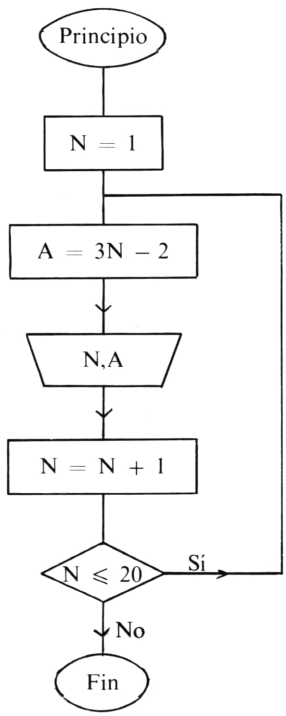
En los ejemplos anteriores hemos visto que el proceso nunca terminaría por sí solo, salvo que al ser ejecutado por un ordenador se llegue al rebosamiento de cifras y éste bloquee los cálculos; hemos tenido que recurrir a una interrupción manual cuando consideramos la observación innecesaria.

La instrucción que vamos a estudiar se corresponde con el bloque de decisión ya estudiado en los organigramas, y es la que permite romper una secuencia de instrucciones «si lo cree conveniente». Por ejemplo, si nos

interesan los 25 primeros términos de la sucesión $a_n = 3n - 2$, podemos construir el siguiente algoritmo:

a) Hacer N igual a 1.
b) Calcular el valor de A del término a_N.
c) Imprimir los valores de N y de A.
d) Aumentar N en una unidad.
e) Si N es menor o igual que 25, volver a *b*.
f) Fin.

El organigrama usa un bloque de decisión:



El programa es:

```
1Ø N=1
2Ø A=3*N-2
3Ø PRINT N,A
4Ø N=N+1
5Ø IF N⩽25 GO TO 2Ø
6Ø END
```

La instrucción que se corresponde con el bloque de decisión es:

```
5Ø IF N⩽25 GO TO 2Ø
```

cuyo significado es: si N es menor o igual que 25, ir a la instrucción número 2Ø, es decir, corresponde exactamente el paso de *e* del algoritmo.

Este programa daría 25 vueltas hasta que, tras mostrar el término 25, el valor de N pasa a ser 26, y al no verificar el condicional toma la salida del Fin y termina el proceso.

La forma general de la instrucción de decisión es:

ne IF (Condición) Instrucción

donde:

ne es un número de etiqueta

— La condición será, por el momento, una relación entre dos valores A y B de uno de los tipos:

$A<B$ $A\leqslant B$ $A>B$ $A\geqslant B$ $A\neq B$ (El signo $\neq$ suele expresarse por $<>$).

— La instrucción de la derecha puede ser un salto incondicional, de la forma GOTO n, como en el ejemplo anterior, o una fórmula; en este último caso, delante de la fórmula escribiremos LET.

Al llegar a esta instrucción del programa, si la expresión del paréntesis es cierta, ejecuta la instrucción de la derecha, y si no, la ignora, pasando a la siguiente instrucción del programa.

EJEMPLO 1

Supongamos que A = 2 y B = 3 y se tienen las instrucciones:

```
3Ø IF (A<B) GO TO 5Ø
...
9Ø IF (A=B) GO TO 15Ø
...
23Ø IF (A<>B) LET X=A-B
```

En la instrucción 3Ø, como el paréntesis es cierto, provoca el salto del control a la instrucción 5Ø.

Al llegar a instrucción 9Ø, como el paréntesis no es cierto, ignora el GO TO 15Ø y pasa el control a la instrucción siguiente de la 9Ø.

En la instrucción 23Ø, al ser cierta la desigualdad, se verifica la instrucción X = A − B.

Tanto el GO TO como el LET, tras un IF, pueden ser sustituidos por la palabra THEN, es decir:

3Ø IF (A<B) GO TO 5Ø es equivalente a 3Ø IF (A<B) THEN 5Ø
23Ø IF (A<>B) LET X=A−B es equivalente a 23Ø IF (A<>B) THEN X=A−B

EJEMPLO 2

Determinar de un grupo de 50 números enteros cuántos son negativos.

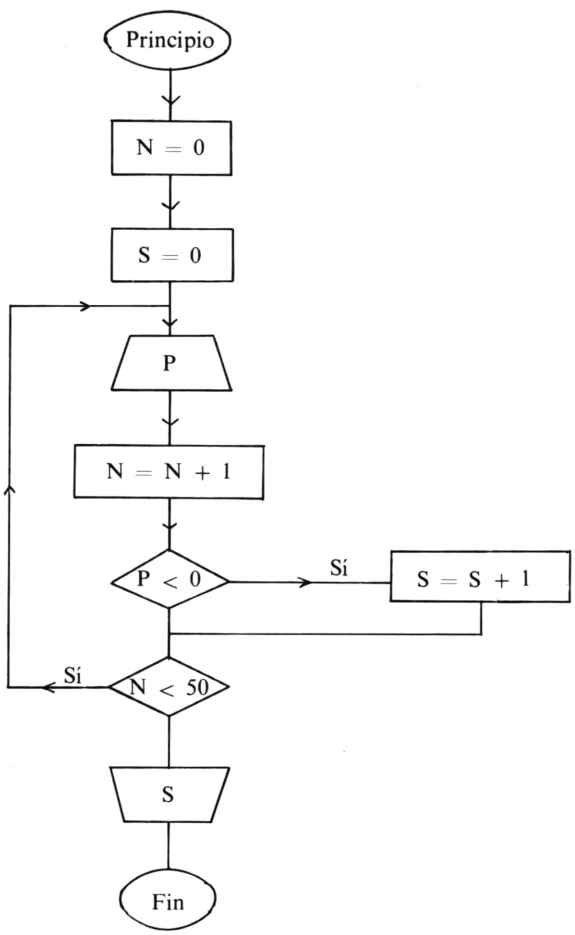
Crearemos la variable S que guardará el número de elementos negativos aparecidos y que al principio ha de valer cero.

La variable N representará el número de elementos que han sido introducidos; al principio ha de valer cero e ir aumentando de uno en uno según vayamos leyendo elementos, y cuando llegue a valer 50 dará el camino para terminar el proceso.

El algoritmo será:

a) Hacer N igual a cero.
b) Hacer S igual a cero.
c) Leer un número P.
d) Aumentar N en una unidad.
e) Si el número leído es negativo, aumentar S en una unidad.
f) Si N es menor que 50, volver al paso *c*.
g) Mostrar el valor de S.
h) Fin.

El organigrama correspondiente es:



De acuerdo con esto se tendría el siguiente programa:

```
1Ø N=Ø
2Ø S=Ø
3Ø INPUT P
4Ø N=N+1
5Ø IF (P<Ø) THEN S=S+1
6Ø IF (N<5Ø) THEN 3Ø
7Ø PRINT S
8Ø END
```

EJEMPLO 3

Resolución de la ecuación general de segundo grado $AX^2 + BX + C = 0$ $(A \neq 0)$.

Representamos por X e Y las partes reales de las soluciones, e I y J las partes imaginarias; es decir, las soluciones son (X, I) e (Y, J). Si las raíces son reales será I = J = 0.

El algoritmo que vamos a seguir es el siguiente:

a) Leer los coeficientes A, B y C.
b) Calcular el discriminante $D = B^2 - 4AC$.
c) Si D es positivo o nulo, ir al paso *f*.
d) Como D es negativo hacer:

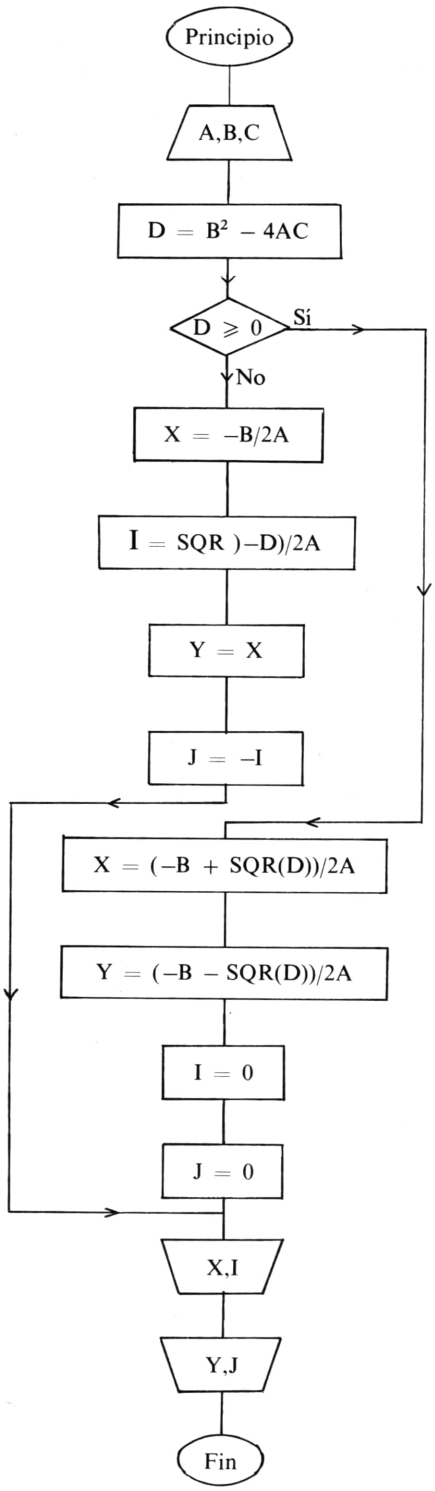
```
X=-B/2A                    I=SQR(-D)/2A
Y=X                        J=-I
```

e) Ir a *g*.
f) Como D no es negativo hacer:

```
X=(-B+SQR(D))/2A           I=Ø
Y=(-B-SQR(D))/2A           J=Ø
```

g) Mostrar resultados X, I e Y, J.
h) Fin.

El organigrama y programa correspondientes son:




```
10 INPUT A,B,C
20 D=B↑2–4*A*C
30 IF D⩾0 THEN 90
40 X=–B/(2*A)
50 I=SQR(–D)/(2*A)
60 Y=X
70 J=–I
80 GO TO 130
90 X=(–B+SQR(D))/(2*A)
100 Y=(–B–SQR(D))/(2*A)
110 I=0
120 J=0
130 PRINT X,I
140 PRINT Y,J
150 END
```

EJEMPLO 4

Obtener los k primeros términos de la sucesión:

1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, ...

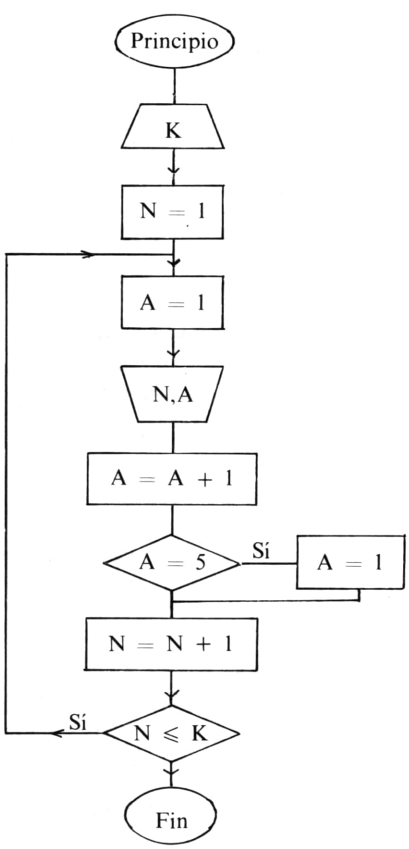
con su número de orden correspondiente.

N va a representar el número de orden del término y A su valor; A tiene ciclos de cuatro, es decir, A toma los valores 1, 2, 3 y 4, pasando a tomar nuevamente el valor 1 en lugar de tomar el valor 5.

Seguiremos, pues, el siguente algoritmo:

a) Leer el valor de K.
b) Inicializar N con el valor 1.
c) Inicializar A con el valor 1.
d) Mostrar N y A.
e) Aumentar A en una unidad.
f) Si A es igual a 5, cambiar este valor por 1.
g) Aumentar N en una unidad.
h) Si N es menor o igual que K, volver al paso *d*.
i) Fin.

El organigrama y programa son:




```
1Ø INPUT K
2Ø N=1
3Ø A=1
4Ø PRINT N,A
5Ø A=A+1
6Ø IF A=5 THEN A=1
7Ø N=N+1
8Ø IF N≤K THEN 4Ø
9Ø END
```

EJERCICIOS

Hcer el análisis y el programa correspondiente a los ejercicios 1, 2 y 3.

1. Hallar el área y el volumen de un ortoedro, conocidas sus tres aristas.

2. Calcular la superficie de una esfera conociendo su radio.

3. Hallar el volumen de una esfera conociendo su área.

4. Indicar el propósito del siguiente programa:

```
1Ø INPUT R,H
2Ø B=π*R↑2
3Ø S=2*B+2*π*R*H
4Ø PRINT B,S
5Ø END
```

5. Estudiar si el siguiente programa es correcto y estudiar su propósito:

```
1Ø INPUT R,H
2Ø V=πR↑2*H
3Ø PRINT V
4Ø END
```

Hacer el análisis y el programa en los ejercicios siguientes:

6. La duración de la trayectoria de un proyectil en el vacío, lanzado a una velocidad v y bajo un ángulo a viene dado por:

$$t = \frac{2v \operatorname{sen} a}{g}$$

siendo g la aceleración de la gavedad (9,8).
Obtener el tiempo t conociendo la velocidad inicial v t el ángulo a.

7. Conseguir mostrar los N primeros enteros positivos.

8. De acuerdo con el ejemplo 7.2 *c* que nos decía que la velocidad de crecimiento estimada del cabello humano era de $5 \cdot 10^{-9}$ m/seg., obtener la longitud del pelo de un individuo de 10 en 10 días hasta que sobrepase los 30 cm, suponiendo que hoy se ha rapado la cabeza.

9. Dada la sucesión $a_n = (2 - 3n^2)/(n^2 + 1)$, obtener los términos:

a_{45}, a_{48}, a_{51}, a_{54}, ...

hasta superar el término 100.

10. Un movimiento rectilíneo responde a la ecuación $e = t^3 - 2t^2 + t - 7$; queremos saber la posición del objeto que se mueve según esa ley, de segundo en segundo, a lo largo de los dos primeros minutos.

11. El extremo de un minutero se desplaza a una velocidad de 10^{-3} m/seg. Determinar la longitud del recorrido en un número de horas dadas.

12. Repetir el ejercicio 11 suponiendo que el tiempo a introducir debe expresarse en horas, minutos y segundos.

13. Obtener la sucesión de sumas parciales hasta S_{75} de:

$$S_i = 1 + 1/2 + 1/2^2 + \ldots + 1/2^i$$

14. Obtener los términos de la sucesión de Fibonacci con su número de orden correspondiente, sabiendo que ésta viene definida por:

$$a_1 = 1 \qquad a_2 = 1 \qquad a_n = a_{n-1} + a_{n-2}$$

15. Un comerciante dispone de tres tipos de cables:

Tipo 1 a 24 ptas. el metro
Tipo 2 a 37 ptas. el metro
Tipo 3 a 52 ptas. el metro

queremos que, dando el tipo y el número de metros deseados, se nos muestre el importe correspondiente.

16. Al lanzar un proyectil bajo un ángulo a, y una velocidad v, la relación entre su desplazamiento horizontal y su altura viene dada por:

$$y = x \operatorname{tg} a - \frac{1}{2} g \frac{x^2}{v^2 \cos a}$$

Hacer una tabla para que una v y a dados, nos dé los valores de la altura «y», al variar x de 10 en 10 metros, hasta que la altura deje de ser positiva.

8

Aplicaciones elementales

8.1. INTRODUCCION

Hasta ahora, los pasos que hemos usado en los algoritmos que resolvían los problemas planteados, responden a uno de los siguientes tipos:

— Entrada de datos.
— Salida.
— Cálculos.
— Saltos incondicionales.
— Decisiones.
— Indicación de fin de proceso.

Cada uno de ellos tiene su representación correspondiente en el organigrama y una instrucción Basic que lo traduce en el programa.

Pues bien, con este reducido número de elementos se puede acceder al análisis y programación de la mayoría de los problemas que en un nivel medio pueden presentarse, tales como los que veremos a continuación y muchos más que irán apareciendo en los siguientes capítulos.

En los cuatro temas seleccionados en un principio: sucesiones, elecciones y ordenaciones, factoriales y divisibilidad, hay procesos muy simples, y otros, que sin ser difíciles, el lector deberá hacer una lectura muy detenida de ellos.

Es necesario hacer notar que la mayoría de los algoritmos y sus organigramas, y por lo tanto de los programas, pueden hacerse más compactos, pero se ha preferido no insistir en estas simplificaciones para no dificultar la comprensión de los esquemas. En este sentido, compactar mucho puede acarrear problemas para entender el proceso, sobre todo cuando éste va a ser leído por otras personas.

Preferimos dejar las posibles mejoras como ejercicio para el lector interesado; hemos procurado que nuestros esquemas no sean tan extensos que se pueda perder la noción de conjunto del problema, ni tan compactos que no puedan ser entendidos.

8.2. SUCESIONES

Como este tema ha sido tratado con frecuencia en los apartados anteriores, incluimos únicamente dos aplicaciones.

1. Obtener la suma de los P primeros términos de la sucesión:

$$0, 3 / 5, 8 / 10, \ldots (n^2 - 1) / n^2 + 1), \ldots$$

mostrando las sumas parciales.

N va a representar el contador de términos que los iniciaremos con el valor 1, y después irá aumentando de uno en uno hasta P.

Usaremos una variable S donde acumularemos las sumas parciales; inicialmente valdrá cero y a ella iremos sumando los términos que vayamos calculando.

El valor de P ha de ser un valor a introducir.

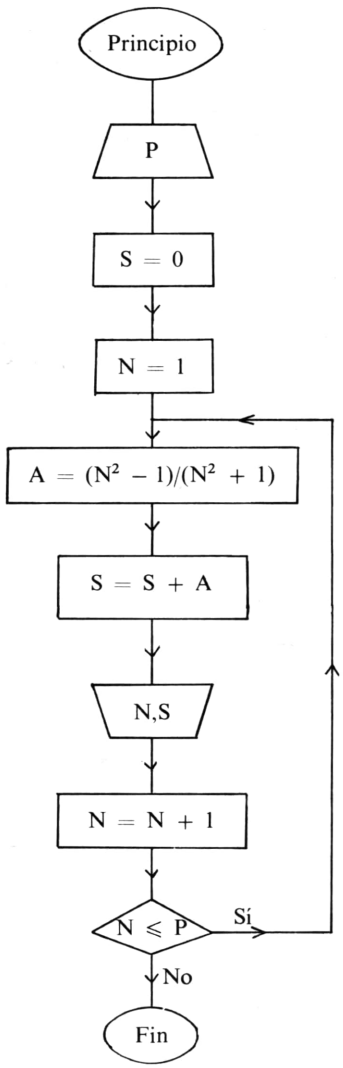
El algoritmo puede ser el siguiente:

a) Dar un valor a P.
b) Hacer S igual a 0.
c) Hacer N igual a 1.
d) Calcular el valor A del término a_N.
e) Añadir a S el valor A del último término calculado.
f) mostrar N y S.
g) Aumentar N en una unidad.
h) Si N es menor o igual que P, volver al paso *d*.
i) Fin.

El último valor de N mostrado será P y la última suma S corresponderá a la suma pedida de los P primeros términos.

Obsérvese que si solamente nos interesase la suma fiñal nos bastaría colocar el paso *f* inmediatamente después del paso *h*.

El organigrama y programa correspondientes a este algoritmo serían los siguientes:




```
1Ø INPUT P
2Ø S=Ø
3Ø N=1
4Ø A=(N↑2-1)/(N↑2+1)
5Ø S=S+A
6Ø PRINT N,S
7Ø N=N+1
8Ø IF (N⩽P) THEN 4Ø
9Ø END
```

2. Obtener los P primeros términos de la sucesión de Fibocacci definida por:

$$a_1 = 1 \quad a_2 = 1 \quad a_n = a_{n-1} + a_{n-2}$$

Observemos que la aparición de los dos primeros términos de la sucesión debe estar aparte del resto, que sigue la ley de recurrencia.

N representará el número de orden del término calculado.

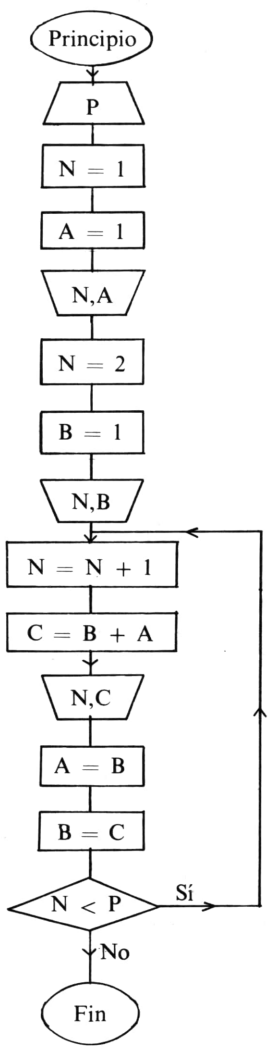
Como cada término necesita, para poder ser calculado, los dos anteriores, es preciso conservar estos valores; usaremos las variables A, B y C con este objetivo, de manera que A = a_{n-2}, B = a_{n-1} y C = A + B; es decir, C será siempre el término a calcular, B el anterior y A el anterior a B;

calculado el valor de C y mostrado haremos A = B y B = C y nuevamente tengo en A y en B los dos últimos términos calculados que me permitirán hallar el siguiente.

El algoritmo podría ser como sigue:

a) Dar un valor a P.
b) Hacer N = 1, A = 1 y mostrarlos.
c) Hacer N = 2, B = 1, y mostrarlos.
d) Aumentar N en una unidad.
e) Calcular el valor C del término $a_N = a_{N-1} + a_{N-2} = B + A$.
f) Mostrar N y C.
g) Asignar a B el valor C de a_N y a A el valor B de a_{N-1}.
h) Si N es menor que P, volver al paso *d*.
i) Fin.

El organigrama y programa serían:




```
1Ø INPUT P
2Ø N=1
3Ø A=1
4Ø PRINT N,A
5Ø N=2
6Ø B=1
7Ø PRINT N,B
8Ø N=N+1
9Ø C=B+A
1ØØ PRINT N,C
11Ø A=B
12Ø B=C
13Ø IF (N<P) THEN 8Ø
14Ø END
```

Este proceso es válido para cualquier sucesión del tipo:

$a_n = a_{n-1} + a_{n-2}$

con sólo variar los valores iniciales de A $= a_1$ y B $= a_2$.

8.3. ELECCIONES Y ORDENACIONES

1. Dado un número natural N, queremos que si es par se calcule su raíz cuadrada, y si es impar que se eleve al cuadrado.

Para ver si N es par o impar tenemos que estudiar el valor del resto de N/2, cuyo valor es R = N − 2 · INT(N/2); si R es cero el número es par, mientras que si vale 1, es impar.

Seguiremos el siguiente algoritmo:

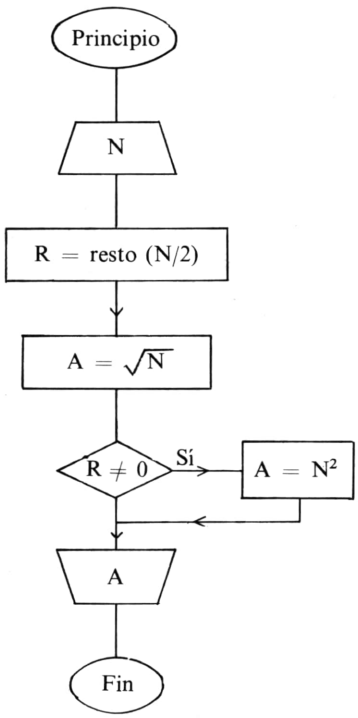
a) Dar un valor a N.
b) Calcular el resto R de la división entera de N entre 2.
c) Si el resto es cero, calcular A = $\sqrt{N}$.
d) Si el resto no es cero, hacer A = N^2.
e) Mostrar el valor de A.
f) Fin.

Si estudiamos detenidamente este algoritmo, nos daremos cuenta que podemos suprimir el condicional del paso *c* y poner simplemente:

c') Hacer A = $\sqrt{N}$.

ya que si el resto es cero, la condición *d* no se verifica y *e* nos muestra el valor de $\sqrt{N}$; si el resto no es cero, la igualdad A = N^2 hace que el valor mostrado sea precisamente el cuadrado de N, que es el requerimiento en este caso.

El organigrama y programa suponiendo el paso *c'* son:




```
1Ø INPUT N
2Ø R=N-2*INT(N/2)
3Ø A=SQR(N)
4Ø IF (R<>Ø) THEN A=N↑2
5Ø PRINT A
6Ø END
```

2. Dado un número natural queremos que:

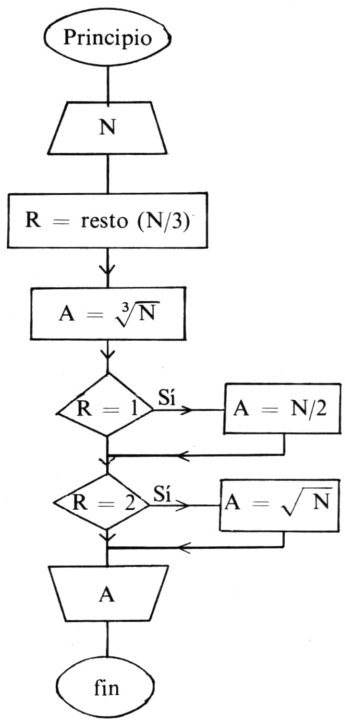
— Si es múltiplo de tres, hallar su raíz cúbica.
— Si es múltiplo de tres, más uno, calcular N/2.
— Si es múltiplo de tres, más dos, hallar su raíz cuadrada.

El ejemplo es análogo al anterior, pero en esta ocasión debemos calcular el resto de N/3 que viene dado por $R = N - 3 \cdot INT(N/3)$; según el valor de R tomaremos una de las tres opciones indicadas. El condicional de la primera lo suprimimos como antes.

Algoritmo:

a) Dar un valor a N.
b) Calcular el resto R de la división entera de N partido por tres.
c) Hacer A igual a la raíz cúbica de N.
d) Si R es uno, hacer A igual a N/2.
e) Si R es dos, hacer A igual a la raíz cuadrada de N.
f) Mostrar A.
g) Fin.

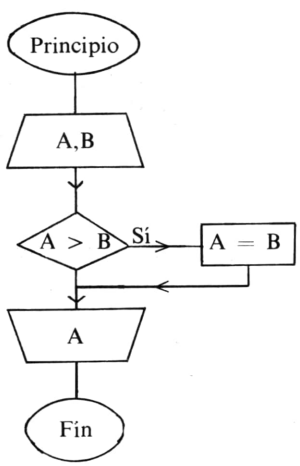
El organigrama y el programa correspondientes son:




```
1Ø INPUT N
2Ø R=N-3*INT(N/3)
3Ø A=N↑(1/3)
4Ø IF (R=1) THEN A=N/2
5Ø IF (R=2) THEN A=SQR(N)
6Ø PRINT A
7Ø END
```

3. Dados dos números A y B, hacer A = mínimo (A, B).
El algoritmo es inmediato:
a) Leer los dos números A y B.
b) Comparar A con B y llamar A al menor de ellos.
c) Mostrar A.
d) Fin.

También su organigrama y su programa son evidentes:



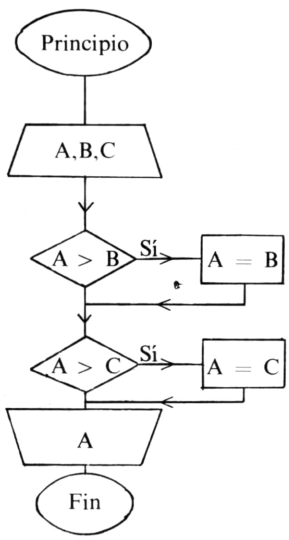

```
1Ø INPUT A,B
2Ø IF (A>B) THEN A=B
3Ø PRINT A
4Ø END
```

4. Buscar el menor de tres números dados A, B y C.

El método que vamos a seguir consistirá en recorrer los tres números en el orden dado, e irnos quedando con el menor que vaya apareciendo, según indica el siguiente algoritmo:

a) Dar los tres valores a A, B y C.
b) Comparar A con B y llamar A al menor.
c) Comparar A (el valor obtenido en el paso *b*) con C y llamar A al menor.
d) Mostrar A.
e) Fin.

organigrama y programa:




```
1Ø INPUT A,B,C
2Ø IF (A>B) THEN A=B
3Ø IF (A>C) THEN A=C
4Ø PRINT A
5Ø END
```

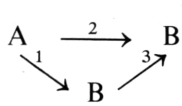
La generalización a más números es inmediata a partir de este ejemplo.

5. Dados dos números A y B queremos hacer A = mínimo (A, B) y B = máximo (A, B).

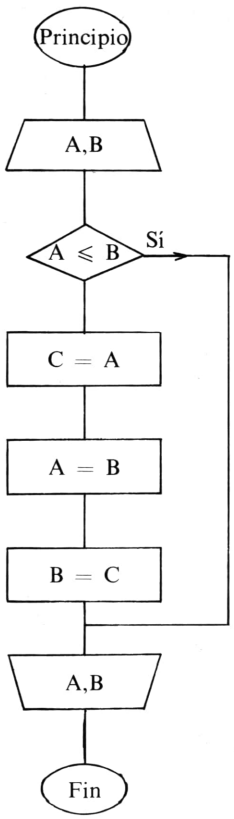
Si fuese A menor o igual que B el problema estaría resuleto, pero si A es mayor que B habría que intercambiar sus valores, según indica el siguiente algoritmo:

a) Dar valores a A y B.
b) Comparar A con B:
  Si A $\leqslant$ B ya están ordenados; ir a *c*.
  Si A > B intercambiar sus valores.
c) Mostrar A y B.
d) Fin.

Para el proceso de intercambio de los valores de A y B necesitamos una variable auxiliar C que almacene uno de estos valores: el valor de A lo ponemos en C, el de B en A y finalmente el de C en B:



Organigrama



Programa

```
1Ø INPUT A,B
2Ø IF (A⩽B) THEN 6Ø
3Ø C=A
4Ø A=B
5Ø B=C
6Ø PRINT A,B
7Ø END
```

6. Dados tres números, ordenarlos de menor a mayor.

El proceso será una repetición del seguido en el ejemplo anterior del cual haremos uso en la descripción del algoritmo. Los tres números serán A, B y C y los ordenaremos como sigue:

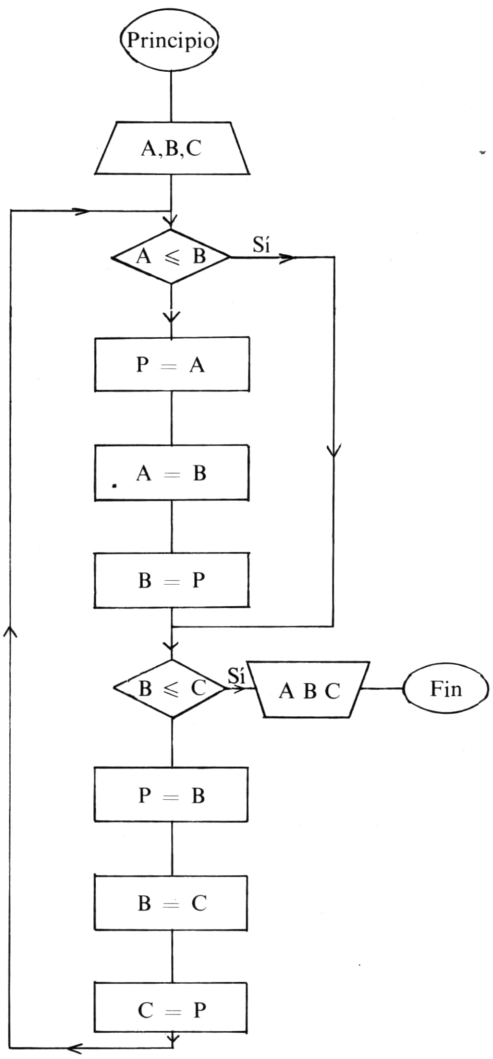
a) leer los números A, B y C.
b) Comparar A con B y ordenarlos.
c) Comparar B con C:
  Si B $\leqslant$ C ya están ordenados; ir a *d*.
  Si B > C intercambiar sus valores y volver al paso b.
d) Mostrar A, B y C.
e) Fin.

Así, por ejemplo, si A = 4, B = 3 y C = 2, los sucesivos valores de A, B y C en los pasos indicados son:

	A	B	C	
a	4	3	2	Datos iniciales.
b	3	4	2	Intercambia A con B.
c	3	2	4	Intercambia B con C.
b	2	3	4	Intercambia A con B.
c	2	3	4	Comprueba que B $\leqslant$ C y envía a d.
d	2	3	4	Salida.

Usaremos la variable P como auxiliar para los intercambios.

Organigrama




```
1Ø INPUT A,B,C
2Ø IF (A⩽B) THEN 6Ø
3Ø P=A
4Ø A=B
5Ø B=P
6Ø IF (B⩽C) THEN 11Ø
7Ø P=B
8Ø B=C
9Ø C=P
1ØØ GO TO 2Ø
11Ø PRINT A,B,C
12Ø END
```

8.4. FACTORIALES

1. Cálculo del factorial de un número natural N.
Para calcular el factorial de un número usaremos la recurrencia:

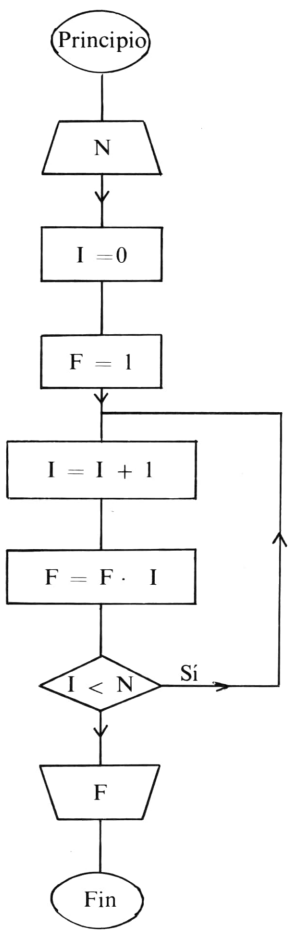
a) Fac(0) = 1.
b) Fac(I) = I · Fac(I − 1).

Usaremos, pues, una variable I que hará de contador desde 0 hasta N y una variable F que irá conteniendo los sucesivos factoriales; I será inicializada a cero y F a uno. El algoritmo es el siguiente:

a) Leer el valor de N.
b) Inicializar I a cero y F a uno.
c) Aumentar I en una unidad.
d) Asignar a F el producto de I por el factorial F anterior.
e) Si I es menor o igual que N, volver a *c*.
f) Mostrar el valor de F.
g) Fin.

Obsérvese que si N es cero el algoritmo es correcto debido a que factorial de cero y de uno coinciden.

Organigrama



Programa

```
1Ø INPUT N
2Ø I=Ø
3Ø F=1
4Ø I=I+1
5Ø F=F*I
6Ø IF (I<N) THEN 4Ø
7Ø PRINT F
8Ø END
```

2. Cálculo del número combinatorio $\binom{M}{N}$.

Este número podría calcularse basándonos en el ejemplo anterior usando la fórmula:

$$\binom{M}{N} = \frac{M!}{N! \cdot (M - N)!}$$

pero supondría un programa de un cálculo muy prolongado innecesario en muchos casos; basta pensar en:

$$\binom{48}{3} = \frac{48!}{3! \cdot 45!} = 17\,296$$

aparte de que muy pocos números combinatorios obtendríamos con todas sus cifras exactas.

Lo haremos usando la siguiente recurrencia:

$$\binom{M}{0} = 1$$

$$\binom{M}{K} = \binom{M}{K - 1} \frac{M - K + 1}{K}$$

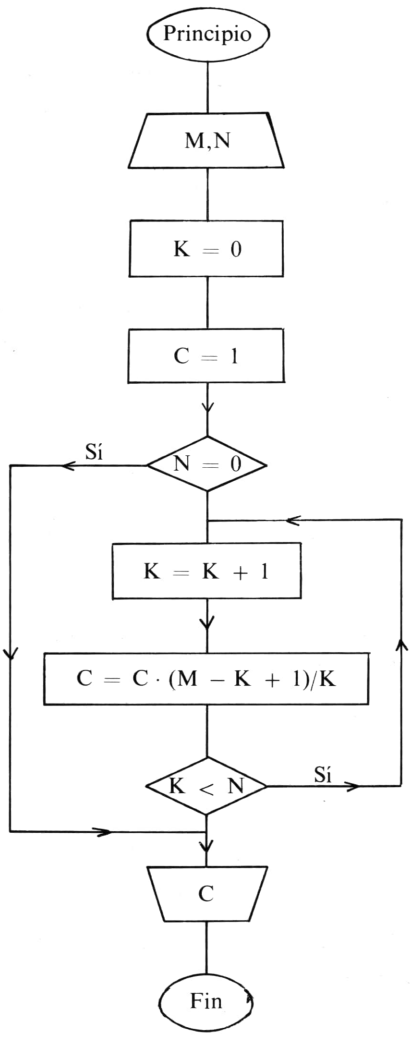
al variar K de cero a N obtendremos finalmente $\binom{M}{N}$; usaremos la variable C para almacenar los sucesivos números combinatorios.

El algoritmo sería el siguiente:

a) Dar valores a M y a N.
b) Inicializar K a cero y C a 1 $\left[\binom{M}{0} = 1\right]$
c) Si N es cero, ir a *g*.
d) Aumentar K en una unidad.
e) Asignar a C el valor del C anterior por $(M - K + 1)/K$.
f) Si K es menor que N, volver a *d*.
g) Mostrar C.
h) Fin.

Como vemos en el algoritmo, el caso en que N sea cero hay que preverlo antes de hacer algún cálculo, pues coincide con el valor inicial de la variable C.

Organigrama



Programa

```
1Ø INPUT M,N
2Ø K=0
3Ø C=1
4Ø IF (N=Ø) THEN 8Ø
5Ø K=K+1
6Ø C=C*(M-K+1)/K
7Ø IF (K<N) THEN 5Ø
8Ø PRINT C
9Ø END
```

Este programa puede mejorarse mucho haciendo uso de la siguiente propiedad de los números combinatorios

$$\binom{M}{N} = \binom{M}{M-N}$$

Con el programa anterior, el cálculo de $\binom{20}{18}$ sería muy largo, mientras que si usamos la propiedad anterior se reduciría a calcular $\binom{20}{2}$ que, evidentemente, simplifica el cálculo.

Esto se consigue añadiendo entre al paso *a* y el *b* del algoritmo el paso *a'*:

a': Si N es mayor que M− N, colocar en lugar de N el valor M − N.

Lo que en el programa significaría añadir la instrucción:

```
15 IF (N>M-N) THEN N=M-N
```

Un ejercicio inmediato a partir de éste, que puede intentar el lector, es la obtención de una fila completa del triangulo de Pascal:

$$\binom{M}{0} \quad \binom{M}{1} \quad \binom{M}{2} \cdots \binom{M}{M}$$

8.5. DIVISIBILIDAD

1. Algoritmo de Euclides para calcular el máximo común divisor de dos números A y B.

Supondremos en primer lugar que A es mayor o igual que B.

Recordemos la siguiente propiedad:

$$\text{m.c.d.}(A,B) = \text{m.c.d.}(B,R)$$

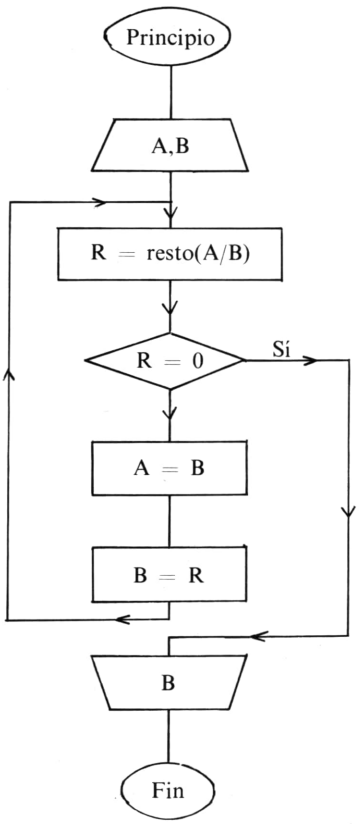
siendo R el resto de la división de A por B; tendremos que ir calculando los restos sucesivos hasta conseguir un resto cero, en cuyo caso el resto anterior es el m.c.d. buscado.

En la igualdad anterior vemos que los valores B y R de la derecha desempeñan la misma función que los valores A y B de la izquierda, así que para no introducir muchas variables distintas, a B y R los volveremos a designar por A y B, respectivamente; seguiremos, pues, el algoritmo siguiente:

a) Dar valores a A y B.
b) Calcular el resto de A dividido por B.
c) Si el resto R es igual a cero, el m.c.d.(A,B) = B; ir a *e*.
d) Si R no es cero, asignar a A el valor de B y a B el de R; volver a *b*.
e) Mostrar B (el último valor de B es el último resto obtenido y por lo tanto el m.c.d.(A,B).
f) Fin.

El organigrama y programa correspondientes a este algoritmo son los siguientes:

Organigrama



Programa

```
1Ø INPUT A,B
2Ø R = A-B*INT(A/B)
3Ø IF (R = Ø) THEN 7Ø
4Ø A = B
5Ø B = R
6Ø GO TO 2Ø
7Ø PRINT B
8Ø END
```

Si eliminamos la restricción de que A sea mayor o igual que B, hay que introducir un paso tal que si A es menor que B intercambie sus valores; en el programa habría que añadir las siguientes instrucciones:

```
12 IF (A<B) THEN 2Ø
14 C=A
16 A=B
18 B=C
```

2. Descomposición de un número natural N en factores primos.

En un primer intento resolveremos el problema simplemente, comprobando si N es múltiplo de 2, 3, 4, 5, ...; si encontramos que N es múltiplo de un número I, seguiremos trabajando con N/I hasta conseguir que N valga 1, con lo que habrán aparecido todos los factores primos del valor N de origen.

En el algoritmo siguiente usamos una variable I que va tomando los valores 2, 3, 4, 5, ...; en primer lugar, se comprueba si N es múltiplo de dos, y, si lo es, ya tenemos un factor primo; dividimos N por dos y con el nuevo valor

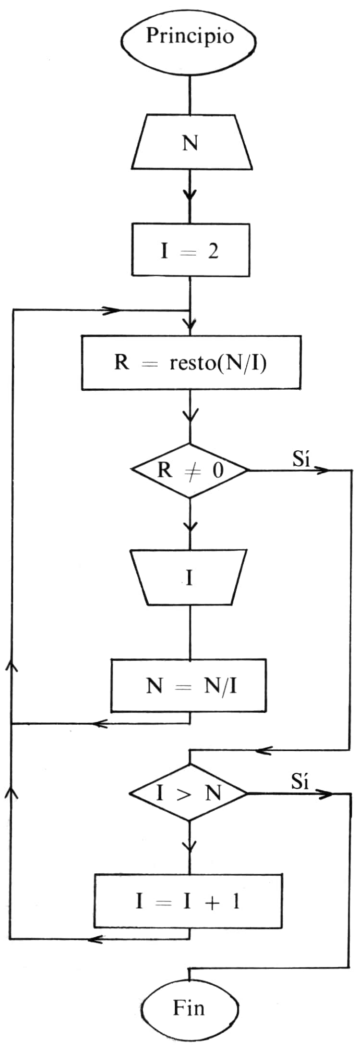
volvemos a comprobar si es múltiplo de dos; si lo es hacemos lo mismo de antes y si no lo es pasaremos a comprobar con el número tres, y así sucesivamente según indicamos a continuación:

a) Dar un valor a N.
b) Hacer I igual a dos.
c) Hallar el resto de N dividido por I.
d) Si el resto R es cero, entonces I es un factor primo; hacer N = N/I y volver a *c*.
e) Si R no es cero:

Si I > N ir a *f*;
Si I ⩽ N aumentar I en una unidad y volver a *c*.

f) Fin.

Organigrama



Programa

```
1Ø INPUT N
2Ø I=2
3Ø R=N-I*INT(N/I)
4Ø IF (R<>Ø) THEN 8Ø
5Ø PRINT I
6Ø N=N/I
7Ø GO TO 3Ø
8Ø IF (I>N) THEN 11Ø
9Ø I=I+1
1ØØ GO TO 3Ø
11Ø END
```

Este algoritmo hace muchas comprobaciones innecesarias, como es el caso de que una vez que hemos llegado a un número que no es múltiplo de dos, comprobar posteriormente si es o no es múltiplo de 4, 6, 8, ..., es realmente un trabajo absurdo; así que, al menos, puede evitarse la comprobación de la divisibilidad por números pares superiores al dos haciendo que I tome los valores 2, 3, 7, 9, 11, ...

Para I = 2 el incremento debe ser de una unidad y para los demás valores de dos unidades, lo que en el programa se traduce en añadir la instrucción:

```
85 IF (I<>2) THEN I=I+1
```

es decir, cuando I es distinto de dos se añade una unidad y junto con la que se añade en la instrucción 9Ø serían las dos exigidas; en otras palabras, cuando I = 2, únicamente surte efecto la instrucción 9Ø pasando I a valer 3; a partir de este momento surten efecto las instrucciones 85 y 9Ø e incrementan los valores de I de dos en dos.

Otra simplificación inmediata viene aclarada a continuación; supongamos que en el programa anterior entramos el número primo N = 1471; el programa comprueba que no es múltiplo de 2, 3, 5, 7, ..., hasta 1471; ahora bien, sabemos que N no puede ser múltiplo de un número mayor que su raíz cuadrada y, por lo tanto, podemos ahorrarnos las comprobaciones para los números comprendidos entre $\sqrt{N}$ y N; para el número 1471 nos ahorraríamos las comprobaciones para todos los números impares comprendidos entre:

$$\sqrt{1471} = 38{,}35 \text{ y } 1471.$$

que corresponden a la mayoría de las comprobaciones.

Esto podemos conseguirlo añadiendo al programa las instrucciones:

```
8Ø IF (I>SQR(N)) THEN 1Ø5
1Ø5 PRINT N
```

cuando la I ha superado la raíz cuadrada de N, nos imprime el valor de N y se termina el programa.

Aún puede intentarse una simplificación mayor teniendo en cuenta que los números primos, salvo el 2 y el 3, son de la forma:

$$6n \pm 1$$

La recíproca no es cierta; no todos los números de esta forma son primos. El ejercicio 12.2.11 hace referencia a esta expresión de los factores primos.

Un ejercicio que puede intentar el lector consiste en, modificando el estudio hecho anteriormente para la descomposición de un número en sus factores primos, resolver el siguiente problema: un número se llama perfecto si es igual a la suma de todos sus divisores y la unidad (se excluye él mismo); por ejemplo, el número 6 es perfecto por ser 6 = 1 + 2 + 3; estudiar un procedimiento que nos diga si un número es o no perfecto.

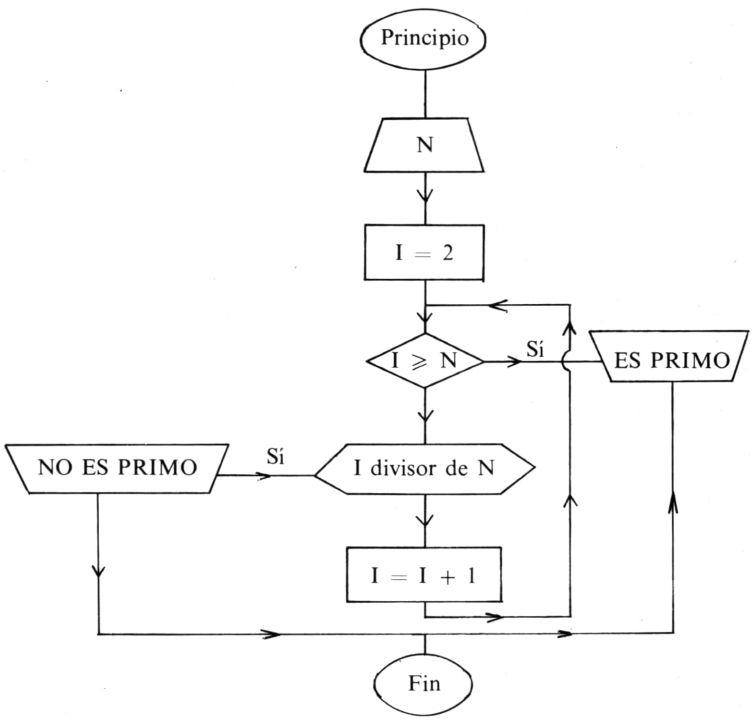
3. Números primos.

Vamos a diseñar un primer lugar un algoritmo que nos diga si un número N mayor que 1 es primo o no; el procedimiento consistirá en comprobar si es múltiplo de 2, 3, 4, ..., N − 1, y si lo es en algún caso, el número N no es primo, mientras que si no es divible por ninguno el número N será primo.

Como antes, la variable I tomará, en principio, los valores 2, 3, 4, ...
El algoritmo es:

a) Dar un valor a N.
b) Hacer I igual a dos.
c) Si I es mayor o igual que N, entonces N es primo; ir a *g*.
d) Si I es divisor de N, entonces N no es primo; ir a *g*.
e) Aumentar I en una unidad.
f) Ir a *c*.
g) Fin.

Organigrama



En el programa deberemos desarrollar un poco más algunos de los pasos especificados; el condicional «Si I es divisor de N» sabemos que exige calcular previamente el resto R y entonces resulta equivalente a decir «Si R = 0». Además, en este programa aparecerá por primera vez una salida que no es un valor numérico, tal como «ES PRIMO» o «NO ES PRIMO»; pues bien, esto lo escribiremos tal cual está incluidas las comillas:

```
ne PRINT "ES PRIMO"
```

y en general se puede poner:

```
ne PRINT "comentario"
```

y el comentario aparecerá tal como figure escrito entre las comillas.

Por lo tanto, el programa será:

```
1Ø INPUT N
2Ø I=2
3Ø IF (I⩾N) THEN 8Ø
4Ø R=N–I*INT(N/I)
5Ø IF (R=Ø) THEN 1ØØ
6Ø I=I+1
7Ø GO TO 3Ø
8Ø PRINT "ES PRIMO"
9Ø GO TO 11Ø
1ØØ PRINT "NO ES PRIMO"
11Ø END
```

Este algoritmo puede mejorarse, como en el caso de la descomposición en factores primos, si en lugar de comparar I con N lo comparamos con raíz cuadrada de N; de la misma forma podemos hacer que I tome los valores 2, 3, 5, 7, ..., aumentando una unidad en el paso del 2 al 3 y dos unidades en los demás casos.

El algoritmo mejorado sería:

a) Dar un valor a N.
b) Hacer I igual a dos.
c) Si I es mayor que $\sqrt{N}$, entonces N es primo; ir a *h*.
d) Si I es divisor de N, entonces N no es primo; ir a *h*.
e) Si I es distinto de dos, aumentar I en otra unidad.
f) Aumentar I en una unidad.
g) Ir a *b*.
h) Fin.

El programa correspondiente a este algoritmo sería:

```
1Ø INPUT N
2Ø I=2
3Ø IF (I>SQR(N)) THEN 8Ø
4Ø R=N–I*INT(N/I)
5Ø IF (R=Ø) THEN 1ØØ
6Ø IF (I<>2) THEN I=I+1
65 I=I+1
7Ø GO TO 3Ø
8Ø PRINT "ES PRIMO"
9Ø GO TO 11Ø
1ØØ PRINT "NO ES PRIMO"
11Ø END
```

Si en lugar de dar un valor a N, en este algoritmo, hacemos que N vaya tomando los valores 2, 3, 4, 5, ... podemos construir una tabla de números primos; el lector interesado hará fácilmente las modificaciones oportunas.

TERCERA PARTE

COMPLEMENTOS DE BASIC. APLICACIONES

9

Expresiones aritméticas y lógicas

9.1. INTRODUCCION

En el capítulo 6 definimos una expresión aritmética como un conjunto de variables, constantes y funciones aritméticas ligadas por los operadores aritméticos; una instrucción aritmética o fórmula era una variable igualada a una expresión aritmética.

Ejemplos:

```
B=2*A-3*X-7
X=-B-SQR(B↑2-4*A*C)
Y=(A+COS(B))/(3-5*SIN(T))
V=24.1367
```

Las propiedades de los cálculos fueron ya establecidas y únicamente había que cuidar de unas determinadas reglas de sintaxis tales como: uso adecuado de paréntesis; no poner productos implícitos; que a la izquierda de una expresión aritmética sólo puede haber una variable, debiendo estar previamente definidas las que figuran a la derecha de la instrucción; etc.

Adoptamos el acuerdo de restringir las constantes a un máximo de once posiciones (incluido punto y signo si los hubiere); si la constante estuviese en forma exponencial, el exponente puede tener dos cifras y el signo precedidos de la letra E.

Ejemplos: de las siguientes constantes

```
-28.356
1473.25919
0.182537E34
54632.4567321E-3
17'391
2673.4
```

La cuarta y quinta no son válidas como constantes: la cuarta por tener más de diez cifras y la quinta por usar coma dicimal que no existe en Basic.

En cuanto a las variables dijimos que serían letras de la A a la Z sin la Ñ; conviene tener en cuenta que para la mayoría de las versiones Basic, las variables pueden ser combinaciones de varias letras y números con tal que la inicial sea una letra.

Así, podrían ser variables:

PEPE A12BC K7 R24Ø

esto es conveniente en programas que necesitan un gran volumen de variables, para designación y mejor identificación de éstas.

No obstante, insistimos que las variables que usaremos en el texto serán de una sola letra y que, en cualquier caso, la sustitución por otro tipo de variables es cosa inmediata.

Hasta ahora nos hemos referido a variables que almacenarán números reales; sin embargo, es de uso frecuente el poder designar las variables enteras de una forma especial; esto se debe a que una variable entera suele ocupar menos memoria que una variable real, y si el ordenador sabe de antemano que una variable solo va a tomar valores enteros reservará menos memoria para ella. La designación de las variables enteras es igual que las variables reales, pero terminan en %; así, por ejemplo, serán variables enteras:

A% B% H%

se usan para economizar la memoria del ordenador cuando hay peligro de que pueda escasear.

EJERCICIOS

Estudiar si son correctas las siguientes instrucciones:

1. C=A+B
2. C=AB
3. A–Z=H
4. √B=2
5. P=(A+B)↑2
6. Q=7²=49
7. 7=P+2
8. –k=5
9. h+3–SINT

Escribir instrucciones aritméticas en Basic que calculen:

10. $y = -2 - \sqrt{b^2 - 4c}$
11. $x = -4 + \sqrt{\text{sen}^2 t + 1}$
12. $v = \text{tg } u - \sqrt{\text{sen } t^2 + 1}$
13. $k = \log v - \sqrt[3]{\text{sen}(1 + t)^2}$
14. $j = \dfrac{a\sqrt{b}}{2} - \dfrac{a^2}{7} \log |a - b|$
15. $x = \dfrac{7 \text{ sen } t}{\pi^{a/4} \cdot \text{tg } (\sqrt{2} \cdot a - b)}$
16. $u = \dfrac{7p^2}{\sqrt[3]{\text{sen}^2(q + 1)}} - \dfrac{2c}{3^2}$

17. $y = \text{tg}\,\dfrac{1 - \text{sen}\ t}{1 + \text{sen}\ t}$

18. $h = \dfrac{\text{tg}^2 - \log p^2}{1 - \ln(q^2 + 3)}$

19. $v = \pi^{-\sqrt{\frac{a}{2bc}}}$

Escribir instrucciones que ejecuten lo siguiente:

20. Multiplicar A por dos y el resultado almacenarlo en B.

21. Restarle B a Z, elevar esto al cuadrado y el resultado almacenarlo en X.

22. Sumar Y con K, hallar a esto la raíz cuadrada, sumar −5 y el resultado almacenarlo en Y.

23. Dividir el logaritmo de X por el coseno de B; elevar esto a la raíz cuadrada de la tangente al cuadrado de T más uno y el resultado almacenarlo en B. La tangente sólo afecta al T y la raíaz cuadrada afecta también al uno.)

9.2. EXPRESIONES LOGICAS

Al igual que existen los operadores aritméticos existen los llamados operadores lógicos; a las expresiones en las que aparecen operadores lógicos (solos o mezclados con los operadores aritméticos) se llaman expresiones lógicas.

Hay dos tipos de operadores lógicos:

a) Operadores de relación.

Son: $=$ $<$ $>$ $\leqslant$ $\geqslant$ $\neq$

$\neq$ normalmente es $<$ $>$ un menor y un mayor enfrentados.

Se aplican de la siguiente forma:

e_1 *operador* e_2

e_1 y e_2 son expresiones aritméticas y el operador uno de los seis anteriores. Ejemplos:

$$A > B \quad 3 \leqslant 7 \quad 3 + 5 > 2 + A \quad 3 \neq 2$$

Al ejecutarse una expresión lógica, el resultado es 1 si es cierta y 0 si es falsa. Ejemplos:

$(3 - 4) < (7 + 5)$	da 1 al ejecutarlo
si $A = 2$ y $B = 3$ $A \neq B$	da 1 al ejecutarlo
si $A = 2$ y $B = 3$ $A = B$	da 0 al ejecutarlo

b) Operadores propiamente lógicos.

Son tres, OR, AND y NOT y coinciden con la disyunción, conjunción y negación lógicas.

Sean e_1 y e_2 dos expresiones con operadores de relación que pueden resultar ciertas o falsas indistintamente.

e_1 OR e_2	el resultado es 0 si ambas son falsas y 1 en los demás casos; coincide con la disyunción lógica.
e_1 AND e_2	el resultado es cierto si e_1 y e_2 lo son simultáneamente y $\emptyset$ en los demás casos; equivale a la conjunción lógica.
NOT e_1	el resultado es 1 si e_1 es falso y $\emptyset$ si es cierto; es la negación lógica.

Ejemplos: si A = 2 y B = 3, se tiene:

(A $\geqslant$ 1) AND (B $\leqslant$ 5)	= 1
(A < 1) AND (B < 5)	= $\emptyset$
(A = 1) OR (B < 4)	= 1
(A = 1) OR (B = 3)	= 1
NOT (A < 5)	= $\emptyset$
NOT (A > 6)	= 1
NOT (B = 2)	= 1

En algunos modelos de máquinas no se admiten estos tres operadores o alguno de ellos, y en otras, aun teniéndolos, su efecto es menos general.

La prioridad en los cálculos de expresiones lógicas es la siguiente:

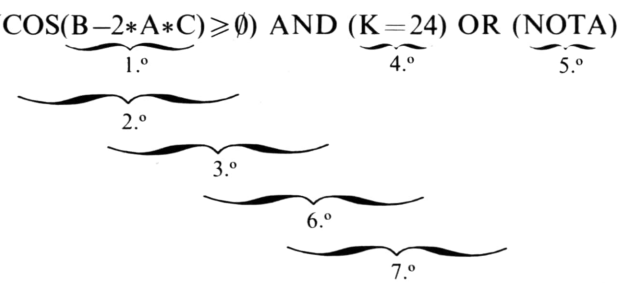
1. Llamada a funciones aritméticas.
2. Signos + y − precedidos del signo de potencia.
3. Exponenciales.
4. Signos + y − unitarios.
5. Multiplicaciones y divisiones * y /.
6. Sumas y restas + y −.
7. Operadores de relación.
8. NOT.
9. AND.
10. OR.

En caso de igual prioridad, el orden de ejecución de las operaciones es de izquierda a derecha; este orden puede ser alterado por el uso de paréntesis, que tienen prioridad sobre el resto y que recomendamos su uso ante la mínima duda.

Si hay varios paréntesis unos dentro de otros, corresponde la preferencia a los más interiores.

EJEMPLO 1

El orden de prioridad en la ejecución de la siguiente instrucción viene dado por:



EJEMPLO 2

Consideremos el intervalo cerrado [3, 7] *y sea X una variable; queremos que si X pertenece a dicho intervalo, la variable Y tome el valor 1 y 0 en caso contrario.*

Esto se puede expresar de una de las dos formas siguientes:

```
Y=(X⩾3) AND (X⩽7)        ó
Y=(X–3)*(X–7)⩽∅
```

EJEMPLO 3

Queremos definir una función F de dos variables X, Y que tome el valor 5 en los puntos pertenecientes a la superficie limitada por la circunferencia $x^2 + y^2 = 4$ *con la parábola* $y = x^2$ *y cero en los demás puntos.*

La expresión que produce este efecto es:

```
F=5*((X↑2+Y↑2⩽4) AND (Y⩾X↑2))
```

EJERCICIOS

1. Sean A, B y C tres variables. Expresar que A valga 1 si está comprendida entre B y C, y 0 en otro caso.

2. Sean P, Q y R variables; si P = 1, escribir para los distintos valores de Q y R las tablas de:

a) P⩽Q.
b) R≠P.
c) R–P⩽Q.
d) (P⩽Q) AND (Q⩽R).
e) (P>Q) OR (P<R).

3. Si A = 2, B = 3 y C = 2, expresar el resultado de:

a) D=(A≤B) AND (B=1) OR (A≤C).
b) F=NOT ((A+B<−3*C)=∅) OR (C−B)<A).
c) H=((B↑2−4*A*C)<∅) OR (A+∅).

4. Sean las variables A, B, C y D; expresar que H es igual a 7 si se cumple al menos una de las condiciones siguientes:

a) Es cero.
b) B es negativo.
c) B = 1 y C = 4 simultáneamente.
d) D es positivo y C negativo al mismo tiempo.

10

Instrucciones elementales

10.1. INTRODUCCION

Hemos visto, aunque en su mínima configuración, las instrucciones:

INPUT	para entrada de datos.
PRINT	para salida.
GO TO n	salto incondicional.
IF	decisiones.
END	fin de programa

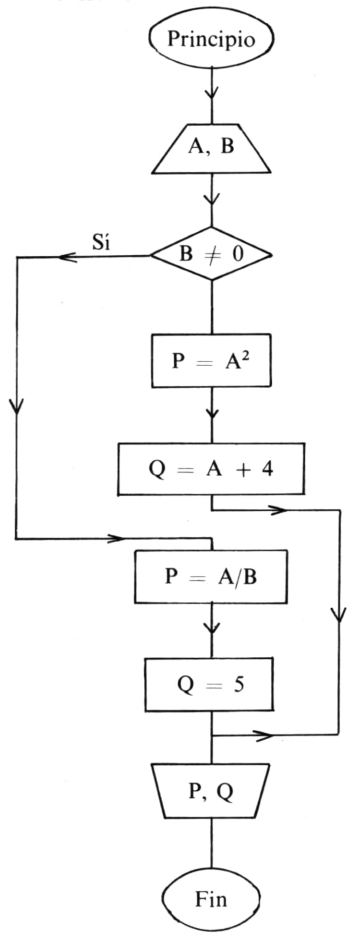
Las cuales volvemos a recordar en el siguiente programa.

EJEMPLO 1

Entrar dos números A y B y queremos que nos muestre los valores de las variables P y Q, sabiendo que:

si $B \neq \emptyset$ es P=A/B Q=5
y
si B=$\emptyset$ es P=A↑2 Q=A+4

Se tiene:




```
1Ø INPUT A,B
2Ø IF (B<>Ø) THEN 6Ø
3Ø P=A↑2
4Ø Q=A+4
5Ø GO TO 8Ø
6Ø P=A/B
7Ø Q=5
8Ø PRINT P,Q
9Ø END
```

Delante de cada fórmula aritmética podríamos poner LET. Por ejemplo:

```
6Ø LET P=A/B
```

y el programa se comportaría de la misma forma (en algunos modelos es obligatorio poner LET). Nosotros únicamente lo usaremos delante de una fórmula que se encuentre en la misma línea de un IF y como alternativa a THEN. Por ejemplo:

```
13Ø IF K=3 LET B=K+2
```

o lo que es equivalente:

```
13Ø IF K=3 THEN B=K+2
```

Los números de etiqueta indicaban el orden de ejecución al ordenador; normalmente los ponemos de 10 en 10 para poder hacer inserciones más tarde, si fuese necesario.

En los siguientes apartados estudiaremos con mayor extensión cada una de las instrucciones anteriores.

10.2. INSTRUCCION INPUT

Consideremos el programa anterior; si intentamos pasarlo en un ordenador escribiremos RUN para llamarlo y al pulsar RETURN aparece en la pantalla un signo de interrogación preguntándonos el valor de A, lo escribimos e introducimos y nos aparece otra interrogación preguntando el valor de B; escribiremos su valor y lo introduciremos para que el ordenador nos dé a continuación los resultados.

En este caso, no es difícil acordarse que la primera interrogación corresponde al valor de A y la segunda al de B, pues se trata de dos valores únicamente; sin embargo, si consideramos la siguiente entrada:

```
1Ø INPUT B,X,K,H,F
```

habría que tener el programa escrito delante de nosotros para saber a qué variable corresponde cada signo de interrogación que aparece.
No obstante, para evitar posibles confusiones, el ordenador permite la aparición de una cadena de caracteres (en lugar del signo de interrogación) que nos indentifique la variable a entrar.

Se vera más claro con el siguiente ejemplo:

```
10 INPUT "VARIABLE A=";A,"VALOR DE B";B
```

en este caso, al llamar el programa introduciendo RUN no aparece un signo de interrogación, sino que aparece en pantalla:

```
VARIABLE A=
```

y si A vale 234 escribiremos este número en pantalla y ésta queda:

```
VARIABLE A=234
```

Lo entramos pulsando RETURN y aparece en pantalla:

```
VALOR DE B
```

y si B ha de valer 56 lo escribimos y queda la pantalla:

```
VALOR DE B56
```

lo entramos y el programa se ejecutaría proporcionándonos los valores de P y Q correspondientes.

Se tiene entonces que antes de cada variable a entrar podemos poner una cadena de caractereres entre comillas separada de la variable por un punto y coma; los caracteres (sin las comillas) aparecerán en pantalla integramente, y a continuación escribimos el valor de la variable. Por esto conviene que el

último carácter de la cadena sea un blanco o el signo igual para que no exista peligro de confusión como en el caso de B, en donde VALOR DE B56 estaría más claro si hubiésemos puesto VALOR DE B=56, es decir, escribir la instrucción de entrada:

```
1Ø INPUT "VARIABLE A=";A, "VALOR DE B=";B
```

Si tenemos varias variables seguimos el mismo procedimiento escribiendo caracteres que dnos permitan identificarla. Por ejemplo, en el caso:

```
1Ø INPUT B,X,K,H,F
```

podemos poner

```
1Ø INPUT"B=";B,"X=";X, "K=";K, "H=";H, "F=";F
```

y nos parece en cada momento la variable a introducir.

Usaremos la palabra «item» para referirnos a vairables o a cadenas de caracteres indistintamente; por ejemplo, el último INPUT consta de 10 items correspondientes a 5 cadenas y 5 variables.

Son correctas las siguientes entradas:

```
22Ø INPUT "COEFICIENTE DE X↑2";X
334 INPUT "CUANTAS FICHAS TOMAS?";F
4ØØ INPUT "NO DES A X EL VALOR UNO;X=";X,"NI A Y EL
    VALOR 7;Y=";Y
420 INPUT "VALOR DE A=";A,"VALOR DE B=";B,"VALOR
    DE C=";C
```

EJERCICIOS

Supongamos que A=2.7, B=54 y C=8 y queremos escribir un INPUT que introduzca estos valores. Analizar lo que ocurre en cada uno de los siguientes casos

1. 1Ø INPUT A,B,C
2. 1Ø INPUT A,B,"C="
3. 1Ø INPUT "A=";B,C
4. 1Ø INPUT "A=";A,B,C
5. 1Ø INPUT "A=";A,"B=";B,C
6. 1Ø INPUT "A=";A,B,"C=";C

En un programa introducimos los valores 7, 4 y 9 en este orden, pero el input al que corresponden pudiera estar incorrecto y no sabemos a qué variables corresponde. Decier los valores de A, B y C en los siguientes casos:

7. 1Ø INPUT A,B,C
8. 1Ø INPUT B,A,C
9. 3Ø INPUT "A=";B,C,A
10. 2Ø INPUT "B=";C,"A=";A,B

11. 32 INPUT "C=";C,"A=";A,"VALOR DE B=";B

12. Escribir para el caso anterior un INPUT que resulte A = 4, B = 7 y C = 9 con cadenas de caracteres que se correspondan con las variables.

13. Estudiar lo que ocurre con la instrucción siguiente:

```
5Ø INPUT A,B,A
```

14. Idem con

```
6Ø INPUT A,A,"A=";B
```

10.3. INTRUCCION PRINT

La intrucción PRINT es de las que depende en mayor grado del modelo particular de ordenador que usemos; damos a continuación lo que pensamos es fundamental y a la vez general de esta instrucción, aunque cada tipo de ordenador incluirá determinadas peculiaridades que habrá que experimentar-las particularmente.

La instrucción PRINT la hemos usado para obtener resultados de un programa; su forma era, por ejemplo:

```
6Ø PRINT X,Y
```

y obteníamos los valores de X y de Y; también puede escribirse:

```
7Ø PRINT 18,27
```

y se obtienen los números 18 y 27.

Al igual que ocurría con la instrucción INPUT, si escribimos:

```
1ØØ PRINT X,B,A,Z,T
```

y nos aparecen los cinco valores en pantalla, puede resultar confuso el acordarse a qué variables corresponden cada uno de los valores aparecidos. Para evitar esta confusión, veremos cómo incluir comentarios que aclaren los resultados obtenidos; no obstante, conviene hacer antes algunas advertencias.

Así como, si en un INPUT hay varias variables, van entrando una tras de otra según el orden de escritura, en salida, todas las variables escritas a la derecha de un PRINT aparecerán simultáneamente en pantalla, por lo que habrá que tener en cuenta la capacidad y distribución de esta para saber exactamente la correspondencia entre los valores obtenidos y las variables cuyos valores hemos solicitado.

Ante esto, es conveniente aclarar que un PRINT puede escribirse separando las variables por comas o por puntos y comas, es decir, puede escribirse:

```
1ØØ PRINT X,B,A,Z,T
ó
1ØØ PRINT X;B;A;Z;T
```

con resultados muy distintos, como veremos.

a) PRINT de varables separadas por comas.

En este caso el ordenador divide cada línea de impresión (de pantalla o impresora, aunque nos referiremos normalmente a la pantalla) en un número fijo de zonas, habitualmente cinco, y sitúa un valor en cada zona; si se piden más variables que zonas la impresión continúa en la línea siguiente.

El número de zonas en que se divide cada línea es fijo para cada modelo; la distribución para nosotros de cada línea será:

zona 1 ↓ zona 2 ↓ zona 3 ↓ zona 4 | zona 5

cinco zonas de once posiciones cada una.

Los números se escriben dentro de su zona a la izquierda, reservando la primera posición para el signo; esta primera posición será un blanco si el número es positivo y el signo menos si el número es negativo.

EJEMPLO 1.

Supongamos que:

```
A=2.3
B=7.8241
C=-74.6
D = 2
E=5.7
F=1234567890
```

y veamos el resultado de los PRINT siguientes:

```
6Ø PRINT A
7Ø PRINT A,B
8Ø PRINT A,B,C,D
9Ø PRINT A,B,C,D,E,F
```

tendríamos:

```
2.3                                                    (6Ø)
2.3           7.8241                                   (7Ø)
2.3           7.8241      -74.6        2               (8Ø)
2.3           7.8241      -74.6        2         5.7
                                                       (9Ø)
1234567890
```

las dos últimas líneas corresponden al PRINT 9Ø que contiene seis variables a imprimir, por lo que sitúa cinco en una línea y la que queda en la línea siguiente.

EJEMPLO 2

Veamos el siguiente.

```
45 PRINT 2,4,3,5,6,7,8,5,4,6,7,9
```

se obtendría:

```
2             4             3             5             6
7             8             5             4             6
7             9
```

Incluimos el subrayado de las líneas para ver claramente la posición de cada número.

b) PRINT de variables separadas por puntos y comas.

En este caso no se divide cada línea de la pantalla en zonas, sino que se imprimen los valores pedidos unos a continuación de otros, hasta terminar la línea para continuar en la siguiente, si fuese necesario; tras imprimir un número se deja un espacio en blanco antes de impirmir el siguiente.

EJEMPLO 3

Considerando los valores del ejemplo 1, veamos el resultado de las siguientes instrucciones:

```
6Ø PRINT A
7Ø PRINT A;B
8Ø PRINT A;B;C;D
9Ø PRINT A;B;C;D;E;F
```

se tendría:

```
2.3                                                          (6Ø)
2.3          7.8241                                          (7Ø)
2.3           78241      -74.6  2                           (8Ø)
2.3          7.8241      -74.6  2      5.7     1234567890   (9Ø)
```

si el número es positivo se reserva el hueco del signo, que es ocupado por un signo menos cuando es negativo; de cualquier forma siempre hay al menos un espacio en blanco entre cada dos números, lo que permite evitar confusiones en la lectura de resultados.

Además de la lista de variables, en un PRINT podemos introducir cadenas de caracteres tal como hicimos en la instrucción INPUT; una cadena de

caracteres actúa en salida como si fuese un dato más, es decir, si la cadena se encuentra en un PRINT con los items separados por comas, ocupa una zona como cualquier otro dato, y si se encuentra en un PRINT con los items separados por puntos y comas, se imprime a continuación del item que le precede y no deja espacio en blanco tras su impresión. Esto se verá mejor con los ejemplos que siguen:

EJEMPLO 4

Supongamos que X = 18, Y = –13 y Z = 4.25; si se tiene:

```
75 PRINT "X=",X,Y,"VARIABLE Z=",Z
```

aparecería lo siguiente

```
X=            18            -13           VARIABLE Z=4.25
```

las cadenas "X=" y "VARIABLE Z=" ocupan una zona de la línea donde son escritas y se sitúan en dicha zona a la izquierda. Si una cadena tiene más de once caracteres, invade la zona o zonas siguientes; el item que venga a continuación se sitúa en la primera zona libre si está separado de la cadena por una coma, mientras que si está separado de dicha cadena por un punto y coma, se imprime a continuación sin dejar ningún espacio en blanco. En algunos modelos de ordenadores que permiten cadenas de longitud muy reducida truncan por la derecha todas las que sean superior al tamaño permitido.

EJEMPLO 5

Si X = 34 se tiene:

```
6 PRINT "EL VALOR DE LA VARIABLE X ES",X
```

el resultado sería:

```
EL VALOR DE LA VARIABLE X ES      34
```

EJEMPLO 6

Sea A=2.7, B=34.6 y C=–8

```
21Ø PRINT "A="; A; "VALOR DE B=";B;"C=";C
```

```
A= 2.7 VALOR DE B= 34.6 C= -8
```

podemos incluir blancos en los comentarios para hacer la salida más clara; así en el caso anterior podríamos escribir:

```
210 PRINT "A=";A;"  VALOR DE B  ";B;"      C=   ";C
```

la salida sería:

```
A=2.7 VALOR DE B=     34.6      C=    -8
```

Para mayor claridad en las expresiones y siempre que exista peligro de confusión usaremos Ø para indicar el cero, distinguiéndolo así de la letra O; del mismo modo usaremos el símbolo ♭ para indicar los espacios en blanco.

EJEMPLO 7

Supongamos que A=34, B=18.7 y C=2; escribamos las salidas correspondientes a las intrucciones 100, 110, 120, 130 y 140

```
1ØØ PRINT "A=",A,"VALOR DE B=",B
```

```
A=            34          VALOR DE B= 18.7
```

```
11Ø PRINT "A=",A"B=,B,"C=",C
```

```
A=            34          B=          18.7        C=
 2
```

```
PRINT "♭♭♭A=♭";A;"♭♭♭B=♭♭";B;"♭♭♭C=♭;C
```

```
  A=  34     B=    18.7  C=  2
```

```
PRINT "EL VALOR DE A ES IGUAL A",A,"B=",B
```

```
EL VALOR DE A ES IGUAL A      34          B=
 18.7
```

```
14Ø PRINT "EL VALOR DE A ES IGUAL A";A;B=;B
```

```
EL VALOR DE A ES IGUAL A 34 B= 18.7
```

Puesto que el tratamiento de una cadena en salida es análogo al de ua variable, podemos escribir un PRINT que contenga sólo cadenas o simplemente un PRINT vacío que provocará una línea en blanco.

EJEMPLO 8

Consideremos las siguientes salidas:

```
15Ø PRINT "TODOS LOS VALORES SON NULOS"
```

```
TODOS LOS VALORES SON NULOS
```

```
16Ø PRINT "TODOS";"LOS";"VALORES";"SON";'NULOS"
```

```
TODOSLOSVALORESSONNULOS
```

```
17Ø PRINT "TODOS","LOS","VALORES","SON","NULOS"
```

```
TODOS       LOS          VALORES       SON          NULOS
```

```
18Ø PRINT
```

Esta instrucción sin lista a imprimir provoca una linea en blanco.

Normalmente cada vez que aparece un PRINT comienza a escribirse una nueva línea, savo que el PRINT anterior termine en coma o en punto y coma, en cuyo caso equivalen a un único PRINT como se indica.

```
PRINT lista1,
PRINT lista2    son equivalentes a PRINT lista1,lista2

PRINT lista1;
PRINT lista 2   son equivalentes a PRINT lista1;lista2
```

EJEMPLO 9

Las dos intrucciones 60 y 70 pueden ser reemplazadas por su equivalente 80

```
6Ø PRINT A,B,
7Ø PRINT "C=",C
8Ø PRINT A,B,"C=",C
```

EJEMPLO 10

Análogamente, las instrucciones 100 y 110 pueden ser sustituidas por 120

```
1ØØ PRINT X;"Y=";
11Ø PRINT Y;"Z=";Z

12Ø PRINT X"Y=";Y;"Z=";Z
```

Realmente, por lo que hemos visto, una coma en un PRINT significa que el siguiente item a escribir va en la siguiente zona libre; un punto y coma

significa que el siguiente item a escribir va inmediatamente a continuación. Pueden combinarse ambos como vemos en el siguiente ejemplo.

EJEMPLO 11

Si X=37.25 e Y=–2, se tienen las siguientes salidas:

```
3Ø PRINT X,,,Y
```

37.25 –2

```
4Ø PRINT "X=";X,"Y=";Y
```

X= 37.25 Y=–2

```
5Ø PRINT ,"TODAS LAS INCOGNITAS VALEN CERO"
```

TODAS LAS INCOGNITAS VALEN CERO

EJERCICIOS

Sea A=2.7, B=3.752, C=1.1721, D=1.43, E=Ø.5, F=Ø.18, G=34 y H=Ø. Escribir la salida correspondiente a los PRINT siguientes:

1. PRINT A,C,D
2. PRINT A;C;E;G
3. PRINT "A=";A;"VALOR DE H=";H
4. PRINT "NO SABEMOS SU VALOR";"ƀA=";A
5. PRINT "NO SABEMOS SU VALOR","ƀA=",A
6. PRINT "ƀƀƀTODOS SON PARESƀƀƀ"
7. PRINT "B–A+F","CALCULO=",
   PRINT E,D
8. PRINT 3;4;5
   PRINT "CONTINUA";56,243
9. PRINT "valor de a=",A,B,"C=";C,"G+H=";G
10. PRINT "C=";C,,,"E=";E
11. Escribir un PRINT que imprima "SOLUCION" en el centro de la línea.

Escribir el PRINT correspondiente a las siguientes salidas, sabiendo que la parte literal corresponde a comentarios y los números a salidas de variables de las anteriormente indicadas.

12.

D VALE 1.43 C NO ES NULO

13.

```
VALOR POSITIVO  F= 0.8                    VOLVAMOS A COMENZAR
```

14.

```
LA LINEA SIGUIENTE ES LA DOS Y B=                              3.752

H=          Ø          DE ES IGUAL A          1.43
```

15. Construir un programa que nos dé las raíces cuadradas de los números comprendidos entre 1 y 100 con un encabezamiento tal como "TABLA DE RAICES CUADRADAS" centrado en la primera línea y a continación la lsta de números con sus raíces cuadradas a la derecha:

a) Según el formato:

```
                    RAIZ CUADRADA DE n.º = su raíz cuadrada
```

b) Según el formato:

```
n.º          raíz          nº          raíz
```

10.4. INSTRUCCION END

Esta instrucción indica que el programa ha terminado; debe ser, por tanto, la última instrucción. Si únicamente hay un programa en memoria, podemos omitirla y no ocurre nada anormal; no obstante, si tenemos dos programas en memoria es fundamental escribir la instrucción END en el primero, para que en caso de ejecutar el primer programa detecte el ordenador donde se termina y no pase automáticamente a la ejecución del segundo tras concluir con el primero.

EJEMPLO 1

A continuación tenemos dos programas; el primero, de la instrucción 10 a la 40, aplica el teorema de Pitágoras calculando la hipotenusa A de un triángulo rectángulo de catetos B y C dados, y el segundo programa, de la instrucción 50 a la 90, que es un contador desde 1 hasta 30.

```
1Ø INPUT B,C
2Ø A=SQR(B↑2+C↑2)
3Ø PRINT A
4Ø END
5Ø I=1
6Ø PRINT I
7Ø I=I+1
8Ø IF (I⩽3Ø) THEN 6Ø
9Ø END
```

Si queremos aplicar el teorema de Pitágoras, llamaremos RUN (ó RUN 1Ø), nos pedirá B y C, imprimirá A parándose seguidamente en la instrucción

4Ø END; si suprimimos la instrucción 4Ø la máquina tras mostrarnos A comenzaría con el contador.

Si lo que deseamos es pasar el contador llamamos RUN 5Ø y van apareciendo los valores de I desde 1 hasta 30, finalizando estos dos programas la instrucción 9Ø puede suprimirse y el contador termina igualmente cuando I alcanza el valor 30.

En resumen, si tenemos varios programas en memoria y queremos correr uno de ellos, basta llamar con RUN *n*, siendo *n* el número de etiqueta correspondiente a la primera instrucción del programa. En los casos de programas intercalados entre otros, la instrucción END resulta imprecindible.

Una instrucción que suele usarse como fin de programa, cuando queremos concluir un proceso en un punto intermedio, es la instrucción STOP, que equivale a GO TO *m*, siendo *m* el número de etiqueta de la instrucción END.

EJEMPLO 2

El siguiente programa lee un número X y si:

$X \geqslant 1$ calcula su raíz cuadrada
$X < 1$ calcula el valor absoluto de X

```
1Ø INPUT X
2ØIF(X< 1)THEN 60
3Ø A=SQR(X)
4Ø PRINT A
5Ø STOP
6Ø A=ABS(X)
7Ø PRINT A
8Ø END
```

La instrucción 50 equivale a 5Ø GO TO 8Ø; de cualquiera de las dos formas el programa pasa el control a la instrucción 80 y se termina.

10.5. INSTRUCCION GO TO

A esta instrucción la llamamos salto incondicional, pues en su forma GO TO *n* provoca, como ya sabemos, una transferencia de control a la instrucción cuyo número de etiqueta es *n*.

EJEMPLO 1

```
.
.
11Ø GO TO 9Ø        el control pasa a la instrucción 9Ø
.
.
18Ø GO TO 3ØØ       el control pasa a la instrucción 3ØØ
.
```

Hasta ahora, en los casos en los que ha aparecido la instrucción GO TO *n*, *n* siempre ha sido una constante; sin embargo, *n* puede ser una variable cuyo valor se corresponda con un número de etiqueta.

EJEMPLO 2

```
.
.
4Ø I=1ØØ
5Ø GO TO I
.
.
```

El control de la variable se transfiere a la instrucción número 100.

EJEMPLO 3

```
.
.
21Ø IF (I≥1) THEN K=4Ø
22Ø IF (I<1) THEN K=1ØØ
23Ø GO TO K
.
.
```

Si I es mayor o igual que uno, el control se transfiere a la instrucción número 40, mientras que si I es menor que 1, el control se transfiere a la instrucción número 100.

el carácter variable de *n* puede usarse, como vemos, para provocar varias ramificaciones.

EJEMPLO 4

Vamos a construir un programa que produzca el siguiente efecto: entrar un número N del 1 al 17; si es un 1, que aparezca en la pantalla LUNES; si es un dos, MARTES, y así sucesivamente, que nos diga el día de la semana que ocupa ese número de orden comenzando por el lunes.

Este programa lo haremos de dos formas, la segunda de las cuales es más selectiva y directa que la primera.

a) Primer procedimiento: Consiste en entrar el número N y colocar siete IF consecutivos para que aquel que sea cierto imprima el día correspondiente.

El programa es evidente en este caso:

```
INPUT N
2Ø IF N=1 THEN PRINT "LUNES"
3Ø IF N=2 THEN PRINT "MARTES"
4Ø IF N=3 THEN PRINT "MIERCOLES"
5Ø IF N=4 THEN PRINT "JUEVES"
6Ø IF N=5 THEN PRINT "VIERNES"
7Ø IF N=6 THEN PRINT "SABADO"
8Ø IF N=7 THEN PRINT "DOMINGO"
9Ø END
```

En este programa vemos que sea cual sea el valor de N, deben recorrerse los siete IF, lo que supone una pérdida de tiempo innecesaria.

Veamos cómo con el segundo procedimiento conseguimos reducir el tiempo de ejecución.

b) Segundo método: consiste en introducir un GO TO *n* con *n* variable, de modo que se ejecute directamente la impresión del día correspondiente al número de entrada, evitando todos los demás.

Para ello fijémonos que escribiendo GO TO 1Ø*N, el control se transfiere a una de las instrucciones 1Ø, 2Ø, 3Ø, 4Ø, 5Ø, 6Ø ó 7Ø, según sea el valor de N; podemos preparar el programa para que la instrucción 1Ø sea PRINT "LUNES", la 2Ø PRINT "MARTES", etc. El programa completo es:

```
 3 INPUT N
 6 GO TO 1Ø*N
1Ø PRINT "LUNES"
15 STOP
2Ø PRINT "MARTES"
25 STOP
3Ø PRINT "MIERCOLES"
55 STOP
4Ø PRINT "JUEVES"
45 STOP
5Ø PRINT "VIERNES"
55 STOP
6Ø PRINT "SABADO"
65 STOP
7Ø PRINT "DOMINGO"
8Ø END
```

En este programa solamente se ejecuta la parte correspondiente al día que se imprime, con el consiguiente ahorro de tiempo.

Una variante de la instrucción GO TO que produce también el efecto anterior, es la instrucción ON-GO TO, cuyo formato es:

ne ON expresión aritmética GO TO n_1, n_2, ... n_k

siendo *ne* el número de etiqueta de esta instrucción; al llegar a ella se evalúa la expresión aritmética si su valor es 1, el control salta a la instrucción n_1, si el valor es 2, el control salta a n_2, y así sucesivamente.

EJEMPLO 5

Para el caso del ejemplo 4 en el segundo procedimiento podemos sustituir la instrucción:

```
6 GO TO 1Ø*N
```

por esta otra:

```
6 ON N GO TO 1Ø,2Ø,3Ø,4Ø,5Ø,6Ø,7Ø
```

Además, de esta forma, los números de la derecha no tienen que verificar ninguna fórmula como antes.

EJERCICIOS

1. Sea N un número entero; sin usar IF conseguir que si

N es un número par, nos imprima "N ES UN NUMERO PAR"
N es impar, "N ES UN NUMERO IMPAR"

Escribir el programa correspondiente.

2. Lanzamos un dado y el resultado obtenido X lo introducimos como dato en un programa y nos resulta que si

X = 1 imprime "ES EL VALOR MAS PEQUEÑO"
X = 2 imprime "PROBABLEMENTE PERDERAS"
X = 3 imprime "HA SALIDO UN TRES"
X = 4, 5, 6 imprime "CASI SEGURO QUE GANAS; SALIO UN ";X

Escribir dicho programa.

10.6. INSTRUCCION IF

Esta instrucción, llamada condicional, la hemos usado hasta ahora en la forma:

IF e_1 operador e_2 GO TO n

ó

IF e_1 operador e_2 LET $v = e_3$

que pueden escribirse sustituyendo GO TO y LET por THEN, como haremos normalmente; es decir:

IF e_1 operador e_2 THEN n

ó

IF e_1 operador e_2 THEN $v = e_3$

e_1, e_2 y e_3 son expresiones aritméticas;
v representa una variable;
n el número de etiqueta de una instrucción;
operador es cualquier operador de relación:

$$> \quad < \quad = \quad \leqslant \quad \geqslant \quad \neq \quad (<>)$$

EJEMPLO 1

```
.
.
.
2Ø IF A-B>2 THEN 12Ø
3Ø
.
.
.
8Ø IF C>25 THEN A=A+1
9Ø
.
.
.
```

la instrucción 20 evalúa A−B y si el resultado es mayor que dos, pasa el control a la instrucción 120, mientras que si no es mayor que dos continúa la instrucción 30; la instrucción 80 si C es mayor que 25 aumenta A en una unidad y si C es menor o igual que 25 se continúa en la instrucción 90 sin variar el valor de A.

El uso más general que puede hacerse de la instrucción IF tiene el siguiente formato:

IF expresión lógica THEN *n*

ó

IF expresión lógica THEN $v = e_3$

La expresión lógica puede incluir operadores de relación y operadores lógicos; cuando sea cierta se ejecuta el salto a la instrucción *n* o la fórmula $v = e_3$, mientras que si la expresión lógica es falsa, el efecto es nulo. En lugar de la fórmula $v = e_3$ puede ponerse un PRINT o un INPUT.

EJEMPLO 2

```
.
.
.
13Ø IF (A<B) AND (C>D) THEN 22Ø
14Ø
.
.
.
19Ø IF (NOT (C=3)) OR (B>Ø) THEN C=B+2
2ØØ
.
```

La instrucción 130, si ocurre que A < B y que C > D pasa el control a la instrucción 220; en otro caso pasa a la instrucción siguiente 140.

La instrucción 190 si no es C = 3 ó si B es positivo hace C= B+2; en otro caso continúa la instrucción 200 sin que se haya alterado el valor de C.

EJEMPLO 4

Tenemos tres números A, B y C y queremos construir tres IF que tengan los siguientes efectos:

Si A<B<C imprimir "ESTAN ORDENADOS DE MENOR A MAYOR"
Si A=B o A=C o B=C imprimir "ALGUNOS SON IGUALES"
Si A>B>C imprimir "ESTAN ORDENADOS DE MAYOR A MENOR"

Serían los tres IF siguientes:

```
4Ø IF (A<B) AND (B<C) THEN PRINT "ESTAN ORDENADOS DE
   MENOR A MAYOR"
5Ø IF (A=B) OR (A=C) OR (B=C) THEN PRINT "ALGUNOS SON
   IGUALES"
6Ø IF (A>B) AND (B>C) THEN PRINT "ESTAN ORDENADOS DE
   MAYOR A MENOR"
```

EJERCICIOS

1. Las variables A y B pueden ser positivas o negativas; hacer X igual al valor absoluto de la menor.

2. Sea un ángulo de N grados; restar sucesivamente 360 grados hasta reducirlo a uno que sea menor que 360 y mayor que 0; el resultado asignarlo a H.

3. Sean X e Y dos números; si la suma de sus cuadrados es menor que 5, que el control se transfiera a la instrucción 320.

4. Si A está comprendido entre dos y tres, que el control pase a la instrucción 400 y en caso contrario, a la 330.

5. Sean P, Q y R tres números; hacer igual al logaritmo del valor obsoluto del mayor de los tres.

6. Sean P, Q y R tres números distintos; hacer X igual al log de la raíz cuadrada del que se encuentre entre los otros dos.

7. Sean A y B dos números; escribir las instrucciones necesarias para que se efectúe un salto a la instrucción:

12Ø si los dos son positivos;
16Ø si sólo uno es positivo;
21Ø si los dos son negativos.

8. Estudiar el efecto de las tres instrucciones consecutivas siguientes:

```
4Ø IF X<1Ø THEN A=A+7
5Ø IF X<5 THEN A=A+5
6Ø IF X<Ø THEN A=A+1
```

9. Dibujar la función cuando X varía desde 1 hasta 8

```
1ØØ IF X<8 THEN F=4
11Ø IF X<5 THEN F=3
12Ø IF X<3 THEN F=1
```

10. Si X es una variable que puede tomar los valores 1, 2 ó 3, estudiar si la instrucción número 21Ø en:

```
21Ø IF X=1 THEN A=3
22Ø IF X=2 THEN A=5
23Ø IF X=3 THEN A=9
```

puede ser sustituida por: 21Ø A=3 ¿Puede hacerse igual con 22Ø-23Ø?

10.7. INSTRUCCION REM

Esta instrucción no es ejecutable; simplemente introduce en el listado del programa comentarios aclaratorios de lo que hace y para lo que sirve, pero estos comentarios no aparecen durante la ejecución.

EJEMPLO 1

Consideremos un contador de 1 a 30:

```
1Ø I=1
2Ø PRINT I
3Ø I=I+1
4Ø IF I<3Ø THEN 2Ø
5Ø END
```

Este programa puede escribirse como sigue:

```
 5 REM PROGRAMA DE UN CONTADOR ENTRE 1 Y 3Ø
 8 REM LA VARAIBLE I HARA DE CONTADOR
1Ø I=1
2Ø PRINT I
3Ø I=I+1
35 REM LA INSTRUCCION 4Ø COMPARA I CON 3Ø
4Ø IF I<3Ø THEN 2Ø
5Ø END
```

El funcionamiento de ambos programas es idéntico; en la ejecución del segundo no aparecen instrucciones 5,8 y 35; están almacenadas interiormente y, como vemos, si accedemos al listado del programa pueden aclarar su contenido.

EJEMPLO 2

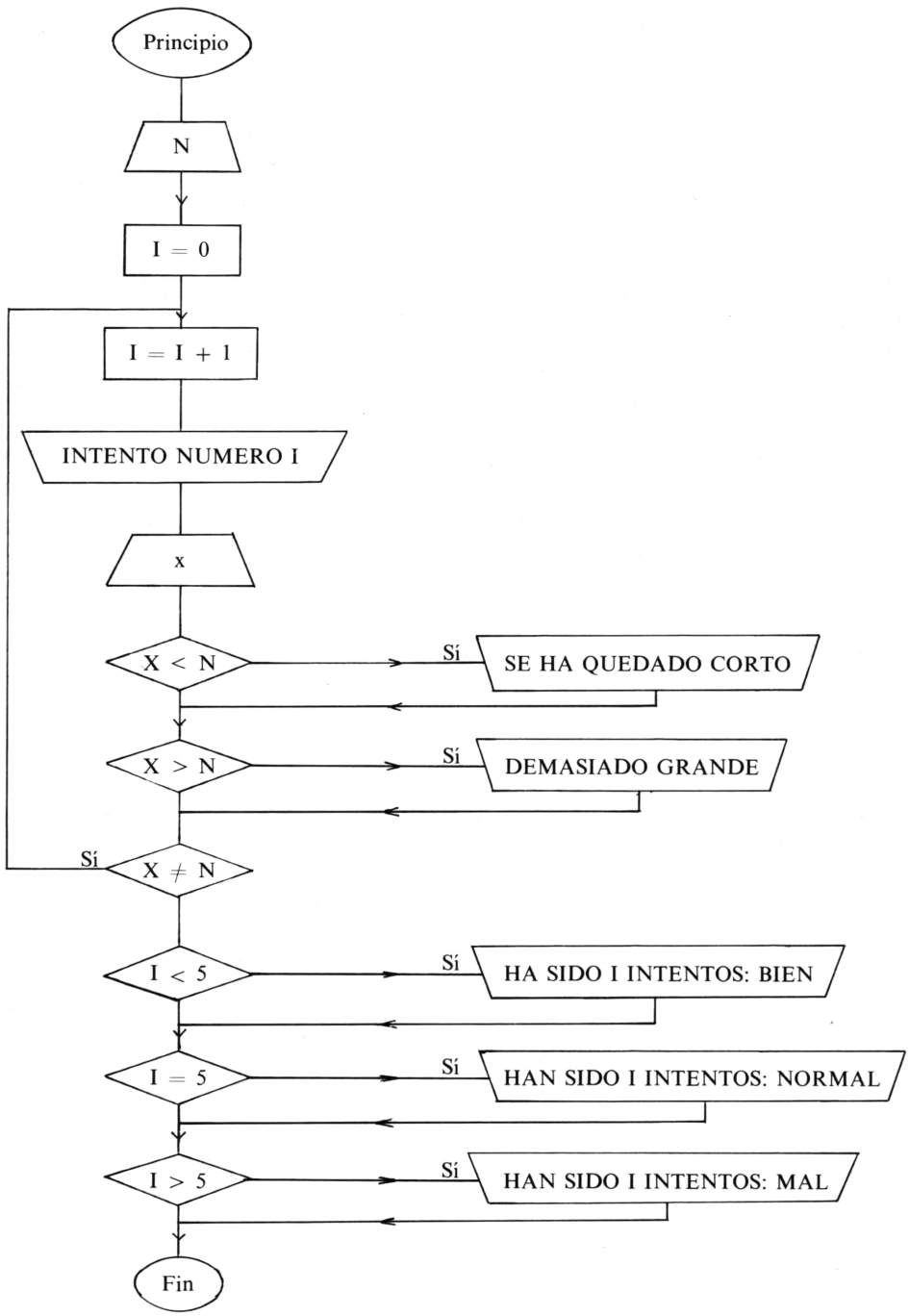
Consideremos el siguente juego: un individuo introduce un número N en la máquina, comprendido entre 1 y 100 (entero); otro individuo debe intentar acertarlo; para ello, debe probar con un número cualquiera entre uno y cien, y la máquina debe responderle si es mayor o menor que el número N buscado. Con esta información hacer un nuevo intento y así sucesivamente.

Cuando al fin lo acierte debe comunicárselo el ordenador y calificarlo de bien, normal o mal, según que haya acertado en menos de cinco intentos, justamente en cinto intentos o en más de cinco, respectivamente. Introducir, además, en el listado del programa comentarios REM que lo aclaren.

El algoritmo puede ser el siguiente:

a) Dar un valor a N entre 1 y 100.
b) Inicializar el contador de intentos I a cero.
c) Aumentar I en una unidad.
d) Entrar un valor de prueba.
e) Si el valor entrado es menor que N imprimir "SE HA QUEDADO CORTO".
f) Si es mayor que N imprimir "NUMERO DEMASIADO GRANDE".
g) Si no hemos acertado, volver a *c*.
h) Si I<5 escribir el número de intentos y la calificación BIEN.
Si I=5 escribir el número de intentos y la calificación NORMAL.
Si I>5 escribir el número de intentos y la calificación MAL.
i) Fin.

Designado por X el valor de cada intento, el organigrama representativo del anterior algoritmo es el siguiente:



El programa con comentarios podría ser el siguiente:

```
1Ø REM PROGRAMA PARA ENCAJAR UN NUMERO N ENTRE 1 Y
   1ØØ
2Ø INPUT "ENTRAR UN NUMERO ENTRE 1 Y 100 N=";N
3Ø REM LA VARIABLE I CONTARA EL NUMERO DE INTENTOS
4Ø I=Ø
5Ø I=I+1
6Ø REM LA VARIABLE X ES EL VALOR DE PRUEBA
7Ø PRINT "INTENTO NUMERO ";I
8Ø INPUT VALOR DE PRUEBA X=";X
9Ø IF (X<N) THEN PRINT "SE HA QUEDADO CORTO"
1ØØ IF (X>N) THE PRINT "DEMASIADO GRANDE"
11Ø IF (X<>N) THEN 5Ø
12Ø IF (I<5) THEN PRINT "HAN SIDO ";I;"INTENTOS: BIEN"
13Ø IF (I=5) THEN PRINT "HAN SIDO ";I;"INTENTOS: NORMAL"
14Ø IF (I>5) THEN PRINT "HAN SIDO ";I;"INTENTOS: MAL"
15Ø END
```

10.8. INSTRUCCION LIST

Esta instrucción no es una sentencia que se pueda incluir en un programa, por lo que no lleva número de etiqueta.

Una vez introducido el programa en el ordenador, si escribimos:

```
LIST 2Ø
```

y pulsamos RETURN, nos aparece en pantalla la línea 20 del programa (si existe) con su contenido.

Si escribimos únicamente:

```
LIST
```

aparece todo el programa en pantalla (listado).

Normalmente, se permite escribir:

```
LIST 2Ø-13Ø
```

y aparecen en pantalla las instrucción cuyos números de etiqueta están comprendidos entre 20 y 130.

La utilidad de esta instrucción es, fundamentalmente, la de revisar los programas introducidos y así poder efectuar las correcciones oportunas, si su funcionamiento no es como quisiéramos, tal como veremos en el siguiente apartado:

10.9. CORRECCIONES A UN PROGRAMA

Una vez introducido un programa en memoria podemos:

a) Borrarlo escribiendo NEW y pulsando RETURN con lo cual se borran todos los programas que hubiesen en memoria.

b) Borrar una línea; por ejemplo, para borrar la línea 4Ø escribimos:

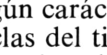
```
4Ø
```

y pulsando RETURN, lo que equivale a introducir la línea 4Ø vacía que machaca la instrucción 4Ø que hubiese antes. El resultado es que en el programa no existe línea 4Ø a partir de ahora.

c) Corregir una línea completa; basta escribir el número de línea a corregir seguida de la instrucción deseada y pulsar RETURN. Por ejemplo, si queremos sustituir la línea 25 A=B–C por K=7, escribiremos sencillamente:

```
25 K=7
```

la línea 25 anterior es machacada y sustituida por ésta.

d) Corregir algún carácter; todas las pantallas disponen de un cursor que moviéndolo con teclas del tipo  permiten borrar y añadir caracteres individuales que permiten hacer correcciones sin tener que escribir la línea completa. Al terminar de hacer correcciones en una línea deberá pulsarse RETURN para que la línea corregida sustituya a la anterior.

Para las correcciones suele usarse la instrucción LIST *n* que trae a la pantalla la línea deseada, lo que es muy útil para programas amplios que no permitan ser visualizados en su conjunto sobre la pantalla.

EJERCICIOS

1. El siguiente programa calcula la hipotenusa A de un triángulo rectángulo conociendo los catetos B y C:

```
1Ø INPUT B,C
2Ø A=SQR(B↑2+C↑2)
3Ø PRINT A
4Ø END
```

Ponerle título con una instrucción REM e incluir algún comentario en la instrucción PRINT.

2. Escribe un programa Basic en el que dando un valor al radio de una circunferencia nos muestre su longitud L y el área A del círculo correspondiente.

Poner título al programa y hacer que los datos L y A aparezcan con comentarios:

a) Sobre una línea, lo más juntos posible.
b) En dos líneas consecutivas.
c) Uno en cada línea con una línea en blanco intermedia.

3. Si A=27.3, B=4 y C=9.Ø18, escribir el efecto de las siguientes instrucciones:

a) `1ØØ PRINT "A=",A;"B=";B,C`
b) `11Ø PRINT "A=";A,"B=";B,"C ES IGUAL A ";C`
c) `12Ø PRINT "VALOR DE A=",A,"VALOR DE B Y C ", B,C`

4. Siendo A, B y C los mismos anteriores, escribir el efecto del siguiente programa:

```
1Ø PRINT A,B,
2Ø PRINT
3Ø PRINT C;A,B;
4Ø PRINT
5Ø PRINT A,B;C
6Ø END
```

5. Un fabricante de cerveza se propone construir botes de medio litro; sabiendo que el cm^2 de superficie cuesta 0,05 ptas., hacer una tabla que nos muestre el importe, en función del radio y de la altura del bote cilíndrico, en tres columnas:

RADIO ALTURA PRECIO

haciendo que el radio varíe desde 1 a 10 cm, de cm en cm.

6. Escribir un programa que nos muestre los números del 1 al 100, sus raíces cuadradas y sus cuadrados, en tres columnas:

NUMERO RAIZ CUADRADA CUADRADO

7. Mediante la inclusión de blancos en el PRINT y comas consecutivas, sin contenido que las separe, conseguir que las siguientes frases aparezcan centradas en la pantalla.

a) RESULTADOS
b) LISTA DE ALUMNOS
c) IMPRESORA AVERIDADA
d) FICHERO NO ENCONTRADO; CAMBIAR CINTA

8. Repetir el ejercicio 7 subrayando la frase.

9. Ingeniarse algún procedimiento para que un programa solamente pueda ejecutarlo el que lo hace.

10. Escribir un programa que traduza notas musicales a literales, sabiendo que la correspondencia es:

Muy deficiente	$0 \leqslant X < 2$
Insuficiente	$2 \leqslant X < 5$
Suficiente	$5 \leqslant X < 6$
Bien	$6 \leqslant X < 7$
Notable	$7 \leqslant X < 8.5$
Sobresaliente	$8.5 \leqslant X < 10$

11. Escribir un programa que nos diga si un numero es o no capicúa.

12. Diseñar un programa para que entrando un día y un mes del año en curso nos diga los días transcurridos desde el 1 de enero.

13. Repetir el ejercicio 12 para un año cualquiera.

14. Escribir un programa al que suministrándole una fecha posterior al 1 de enero de 1901 nos suministre el número de días transcurridos desde entonces.

15. Haciendo uso del programa del ejercicio 14, conseguir modificarlo para que suministrándole una fecha posterior al 1 de enero de 1901 nos muestre el día de la semana al que corresponde.

16. Una ventana rectangular ha de tener 1 m^2 de luz; haz una tabla en la que variando su anchura desde 20 cm hasta 5m, de 10 en 10 cm, nos muestre en tres columnas:

ANCHURA ALTURA PERIMETRO

17. Sabiendo que en el mercado de divisas el marco alemán se cotiza a 52,8 ptas., escribir un sprograma que permita el intercambio entre pesetas y marcos preguntando qué cantidad se desea cambiar y nos comunique el importe en la otra moneda, tanto en un sentido como en el otro.

18. Una casa de alquiler de coches ofrece dos posibilidades:

a) 8 000 ptas. al día y kilometraje ilimitado.
b) 3 000 ptas. al día más 15 ptas. por kilómetro.

Construir una tabla que nos dé el importe en tres columnas en la forma:

KM. PRIMERA FORMULA SEGUNDA FORMULA

haciendo variar los km de 25 en 25, desde 50 hasta 100 km.

19. Sobre el texto A de una barra de 2 m se encuentra un peso de 40 kg. Hacer una tabla con la fuerza que debemos aplicar sobre el otro extremo B para levantar dicho peso, haciendo variar el punto de apoyo de 10 en 10 cm en la dirección de A hacia B.

11

Ciclos FOR-NEXT

11.1. CICLOS SIMPLES

Los ciclos o bucles FOR-NEXT junto con IF y GO TO constituyen las tres instrucciones de control más importantes del lenguaje Basic.

Consideremos un programa que nos muestre los 30 primeros términos de la sucesión definida por $a_n = 2n - 1$ (números impares). Podemos escribir:

```
1Ø I=1
2Ø A=2*I-1
3Ø PRINT A
4Ø I=I+1
5Ø IF I⩽3Ø THEN 2Ø
6Ø END
```

La intrucción 10 inicializa la variable I que va a servir de contador de vueltas; la instrucción 4Ø aumenta el valor de I en una unidad cada vuelta; la instrucción 5Ø es el control para cuando se completen las 30 vueltas se termine el programa.

Hemos usado en múltiples ocasiones instrucciones de control de este tipo combinando un IF y un GO TO para ejecutar repetidas veces un conjunto de instrucciones; no obstante, si sabemos de antemano el número de vueltas que ha de dar el bucle (30 en el caso anterior) podemos usar la instrucció FOR-NEXT, cuya forma es:

etiqueta FOR variable = valor inicial TO valor final

. . . . . }
. . . . . } instrucciones a repetir
. . . . . }

etiqueta NEXT variable

EJEMPLO 1

```
3Ø FOR I=1 TO 3Ø
   . . . . . .
   . . . . . .
   . . . . . .
9Ø NEXT I
```

Al llegar el programa a la instrucción 30, I toma el valor 1 y seguidamente, ejecuta todas las instrucciones hasta llegar al NEXT I; en este momento, salta el control nuevamente a la instrucción 30 tomando I el valor 2, y así sucesivamente hasta que I toma el valor 30; cuando I vale 30 se ejecutan otra vez todas las instrucciones hasta llegar al NEXT I, que en esta ocasión no devolverá el control a la instrucción 30, prosiguiendo entonces la ejecución de las instrucciones que vengan a continuación de la 90.

Así, un programa equivalente que nos imprime los 30 primeros números impares sería el siguiente:

```
1Ø FOR I=1 TO 3Ø
2Ø A=2*I-1
3Ø PRINT A
4Ø NEXT I
5Ø END
```

Como es lógico, el valor inicial de la variable ha de ser menor o igual que el valor final, ya que en caso contrario daría error.

Los valores inicial y final no han de ser necesariamente constantes; estos valores pueden venir dados por variables o expresiones que puedan ser evaluadas; en cualquier caso, estos valores, así como el de la variable que hace de contador, no deben ser alterados en el interior del bucle.

EJEMPLO 2

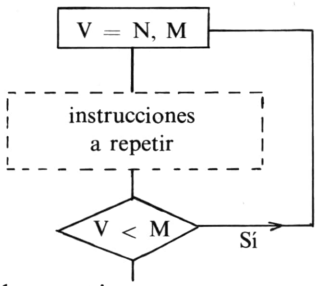
Vamos a construir una tabla con las raíces cuadradas de los números comprendidos entre N y M, siendo $N \leqslant M$.

```
1Ø INPUT N,M
2Ø FOR V=N TO M
3Ø X=SQR(V)
4Ø PRINT X
5Ø NEXT V
6Ø END
```

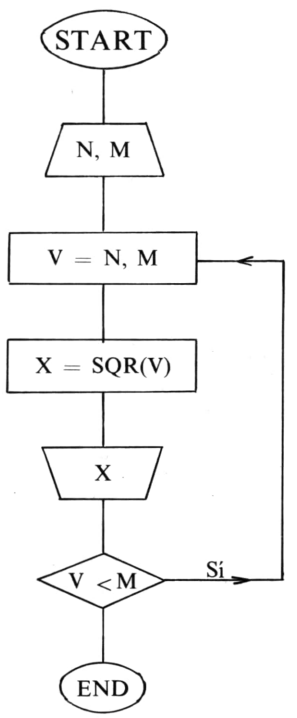
En este caso, la tabla depende de los valores que demos a N y M.

El primer valor a imprimir sería a_N, el siguiente a_{N+1}, y así sucesivamente hasta a_M.

La representación de un organigrama será:



así, en el caso del ejemplo dos, se tiene:



Observar que el FOR comienza con variable = valor inicial, valor final y acaba comparando la variable con el valor final; siempre que el valor de la variable sea inferior al valor final hay un salto al recuadro donde comienza el FOR (no antes, ya que esto significaría inicializar nuevamente la variable).

EJEMPLO 3

El siguiente programa numera diez líneas de 1 a 10 en la forma:

```
LINEA NUMERO    número

1Ø FOR I=1 TO 1Ø
2Ø PRINT "LINEA NUMERO    ";I
3Ø NEXT I
4Ø END
```

En los ejemplos anteriores, la variable del FOR va tomando valores consecutivos, es decir, la variable se incrementa en una unidad en cada vuelta; sin embargo, el incremento de la variable puede ser cualquier valor.

El formato más general sería:

etiqueta FOR variable = valor inicial TO valor final STEP incremento

. . . . . .
. . . . . instrucciones a repetir
. . . . . .

etiqueta NEXT variable

EJEMPLO 4

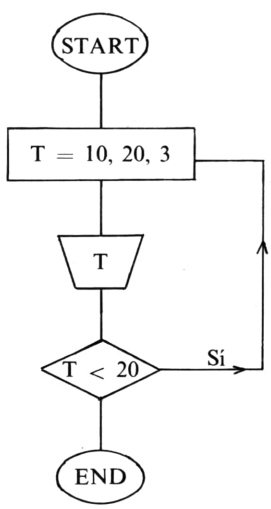
```
6Ø FOR T=1Ø TO 2Ø STEP 3
7Ø PRINT T
8Ø NEXT T
9Ø END
```

Imprimiría:

```
1Ø    corresponde al valor inicial
13    corresponde al valor inicial más 3
16    corresponde al valor anterior más 3
19    corresponde al valor anterior más 3
```

el siguiente a imprimir sería el 21 que como supera al 20, que es el valor final, no se imprime.

Cuando el incremento de la variable sea distinto de uno es obligatorio indicarlo de esta forma; en el organigrama añadiremos este nuevo valor a continuación del valor final y separándolos con una coma. Así, para el ejemplo 4, tendríamos:



EJEMPLO 5

El incremento puede ser negativo como ocurre en el caso siguiente:

```
12Ø FOR J=15 TO 7 STEP-2
13Ø PRINT J
14Ø NEXT J
15Ø END
```

La impresión sería:

```
15    valor de J en la primera vuelta
13    valor de J en la segunda vuelta
11    valor de J en la tercera vuelta
9     valor de J en la cuarta vuelta
7     valor de J en la quinta vuelta
```

Si el incremento de la variable es negativo, el valor inicial de la variable ha de ser mayor o igual al valor final, y en el organigrama lo especificaremos así (en lugar de menor o igual).

EJEMPLO 6

El incremento de la variable puede tomar valores decimales:

```
2Ø FOR K=1 TO 3 STEP Ø.5
3Ø PRINT K
4Ø NEXT K
5Ø END
```

Resultarían los números siguientes:

```
1
1.5
2
2.5
3
```

Evidentemente, el incremento de la variable no puede ser cero.

Hay dos formas de salir de un bucle FOR; la primera esla salida normal cuando la variable alcanza el valor final y concluye el efecto del FOR. La segunda es usando un GO TO *n* siendo *n* una instrucción fuera del bucle.

La entrada, en cambio, únicamente puede hacerse por el principio del FOR y nunca directamente a su interior por medio de una instrucción GO TO.

Los bucles FOR-NEXT aumentarán su potencial enormemente cuando tratemos en el tema siguiente las variables subindicadas.

Muchos de los ejemplos desarrollados en anteriores capítulos pueden simplificarse con el uso de esta instrucción.

EJEMPLO 7

Factorial de un número

```
 5 REM CALCULO FACTORIAL DE UN NUMERO N
1Ø INPUT "N=";N
2Ø F=1
3Ø IF N=Ø THEN 7Ø
4Ø FOR I=1 TO N
5Ø F=I*F
6Ø NEXT I
7Ø PRINT "FACTORIAL DE ";N;" = ";F
8Ø END
```

La instrucción 30 evita el bucle si N vale 0.

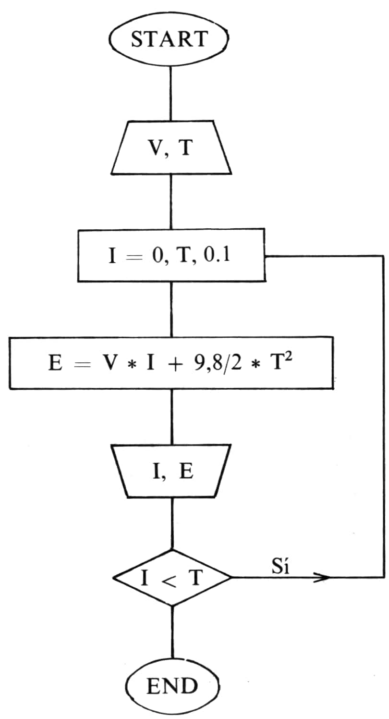
EJEMPLO 8

Sabemos que al lanzar un cuerpo al vacío el espacio que recorre viene dado por la fórmula:

$$e = v_0 t + \frac{1}{2} g t^2$$

donde e es el espacio recorrido en el tiempo t; v_0 es la velocidad inicial y $g = 9{,}8$ es la aceleración de la gravedad.

Vamos a construir un programa que nos de una tabla con los espacios recorridos cada décima de segundo, durante los T primeros segundos para una velocidad inicial V dada.



```
 5 REM TABLA ESPACIO-TIEMPO
1Ø INPUT "V=";V,"T=";T
2Ø FOR I=Ø TO T STEP Ø=1
3Ø E=V* +9.8/2*T↑2
4Ø PRINT "TIEMPO=";I;" ESPACIO=";E
5Ø NEXT I
6Ø END
```

11.2. CICLOS ANIDADOS

Los ciclos FOR-NEXT pueden estar contenidos unos en otros como en el caso:

```
      ...
3Ø FOR I=1 TO 2Ø
      ...
      ...
6Ø FOR J=17 TO 3 STEP -2
      ...
      ...
14Ø NEXT J
      ...
      ...
21Ø NEXT I
      ...
```

cada bucle debe tener su propia variable de control y el orden de aparición de los NEXT debe ser el inverso de los FOR.

El proceso de ejecución sería el siguiente:

a) En 30, I toma el valor 1.
b) Ejecución de las instrucciones entre 30 y 60.
c) El bucle FOR J se ejecuta completamente hasta que J = 3.
d) Ejecución de las instrucciónes entre 140 y 210.
e) Al llegar a 210, si I es menor o igual que 20, se incrementa en una unidad y vuelve al apartado b con el nuevo valor de I.

EJEMPLO 1

Estudiemos el siguiente programa:

```
1Ø FOR I=3 TO 8
2Ø FOR K=1 TO 3
3Ø A=I+K
4Ø PRINT I;"  +  ";K;"  =  ";A
5Ø NEXT K
6Ø NEXT I
7Ø END
```

El proceso sería:

a) En 10, I toma el valor 3.
b) Ejecución de 20 a 50 tres veces, según K = 1, 2 ó 3 sucesivamente.
c) Si I es menor o igual que 8, incrementar I en una unidad y volver al apartado b.

Tendríamos la siguiente impresión:

```
3 + 1 = 4 )
3 + 2 = 5 }  Para I = 3   K toma los valores 1, 2 y 3
3 + 3 = 6 )

4 + 1 = 5 )
4 + 2 = 6 }  Para I = 4   K toma los valores 1, 2 y 3
4 + 3 = 7 )

5 + 1 = 6 )
5 + 2 = 7 }  Para I = 5   K toma los valores 1, 2 y 3
5 + 3 = 8 )

6 + 1 = 7 )
6 + 2 = 8 }  Para I = 6   K toma los valores 1, 2 y 3
6 + 3 = 9 )

7 + 1 = 8 )
7 + 2 = 9 }  Para I = 7   K toma los valores 1, 2 y 3
7 + 3 =10 )

8 + 1 = 9 )
8 + 2 =10 }  Para I = 8   K toma los valores 1, 2 y 3
8 + 3 =11 )
```

EJEMPLO 2

Sabemos que las variaciones con repetición de los números 1, 2 y 3 son $3^3 = 27$*; el siguiente programa con tres ciclos anidados nos imprime estas variaciones como números de tres cifras en columna:*

```
1Ø FOR I=1 TO 3
2Ø FOR J=1 TO 3
3Ø FOR K=1 TO 3
4Ø PRINT I;J;K
5Ø NEXT K
6Ø NEXT J
7Ø NEXT I
8Ø END
```

Primer nivel · Segundo nivel · Tercer nivel

El resultado sería:

```
111 )
112 }  I = 1    J = 1    K = 1, 2, 3
113 )

121 )
122 }  I = 1    J = 2    K = 1, 2, 3
123 )

131 )
132 }  I = 1    J = 3    K = 1, 2, 3
133 )

211 )
... }  I = 2    J = 1    K = 1, 2, 3
... )

... )
... }  I = 3    J = 3    K = 1, 2, 3
333 )
```

Obsérvese que los índices de los niveles más interiores son los que varían más rapidamente.

El resultado puede obtenerse en tres columnas sin más que añadir:

```
40 PRINT I;J;K;"   ";
55 PRINT
```

Obtendríamos lo siguiente:

```
111     112     113
121     122     123
131     132     133
211     212     213
221     222     223
231     232     233
311     312     313
321     322     323
331     332     333
```

Si sobre el programa original añadimos:

```
40 PRINT I;J;K;"   ";
65 PRINT
```

resultaría la siguiente impresión:

```
111   112   113   121   122   123   131   132   133
211   212   213   221   222   223   231   232   233
311   312   313   321   322   323   331   332   333
```

Se debe, como antes, a que la instrucción 40 al terminar en punto y coma obliga a que el siguiente dato se escriba a continuación sobre la misma línea; pero al escribir todos los valores cuando I = 1 (nueve en total) se encuentra con la instrucción 65 PRINT que no imprime nada y provoca el salto a la línea siguiente para el siguiente dato.

EJEMPLO 3

Estudiemos el siguiente programa:

```
1Ø FOR I=1 TO 1Ø
2Ø FOR J=1 TO I
3Ø PRINT "$";
4Ø NEXT J
45 PRINT
5Ø NEXT I
6Ø END
```

El proceso es el siguiente:

a) En 10 I toma el valor 1.
b) El ciclo de 20 a 40 se ejecuta I veces, lo que provoca imprimir I signos $ sobre el mismo renglón por terminar el PRINT en ";".
c) La instrucción 45 provoca que la siguiente impresión ocurra sobre la línea siguiente.
d) I aumenta en una unidad, si no ha superado 10, y vuelve a *b*.

El resultado es:

```
$
$$
$$$
$$$$
$$$$$
$$$$$$
$$$$$$$
$$$$$$$$
$$$$$$$$$
$$$$$$$$$$
```

en el primer renglón la I vale 1, por lo que se imprime un único $; en el segundo, la I vale 2 y se imprimen dos signos $, y así sucesivamente hasta que I vale 10 y se imprimen diez dólares.

EJERCICIOS

1. En una quiniela de cinco partidos escribir un programa que imprima todos los resultados posibles.

2. Calcular la suma de las raíces cuadradas de los números impares mayores que 24 y menores que 100, usando la sentencia FOR.

3. Estudiar cuáles de los siguientes ciclos son correctos:

a)
```
30 FOR H=56 TO 89
   . . .
90 NEXT H
```

b)
```
45 FOR D=56 TO -45 STEP -7
   . . .
70 NEXT D
```

c)
```
110 FOR T=45 TO 76 STEP -4
    . . .
200 NEXT T
```

d)
```
 30 G=7
 40 F=2
500 FOR I=G-F TO G*F STEP 0.5
    . . .
310 NEXT I
```

e)
```
500 FOR L=10 TO 34
    . . .
300 NEXT L
```

f)
```
 35 FOR U=34 TO 89 STEP 6
    . . .
 60 NEXT I
```

g)
```
 20 FOR S=5 TO 100 STEP 2
 30 FOR H=5 TO -5 STEP -1
    . . .
240 NEXT S
250 NEXT H
```

h)
```
 50 L=4
 60 P=9
 70 FOR I=L TO P+L
 80 FOR K=1 TO I
 90 FOR B=100 TO K STEP -1
    . . .
450 NEXT B
460 NEXT K
470 NEXT I
```

4. Estudiar el resultado del siguiente programa:

```
1Ø FOR I=1 TO 1Ø
2Ø FOR J=1 TO 10
3Ø IF I=J THEN 5Ø
4Ø NEXT J
5Ø PRINT "I Y J SON IGUALES A";I
6Ø NEXT I
7Ø END
```

5. Sabiendo que un ordenador trabaja con ángulos en radianes, hacer un programa que construya una tabla con los senos de 30 y 45 grados, de minuto en minuto.

6. Escribir programas que provoquen las siguientes salidas:

a)

```
          $
         $$
        $$$
       $$$$
      $$$$$
     $$$$$$
    $$$$$$$
   $$$$$$$$
  $$$$$$$$$
```

b)

```
          $
         $$$
        $$$$$
       $$$$$$$
      $$$$$$$$$
     $$$$$$$$$$$
    $$$$$$$$$$$$$
   $$$$$$$$$$$$$$$
  $$$$$$$$$$$$$$$$$
```

c)

```
$                   $
$$                 $$
$$$               $$$
$$$$             $$$$
$$$$$           $$$$$
$$$$$$         $$$$$$
$$$$$$$       $$$$$$$
$$$$$$$$     $$$$$$$$
$$$$$$$$$   $$$$$$$$$
$$$$$$$$$$$$$$$$$$$$
```

7. Repetir el ejercicio 10.9.5 usando un ciclo FOR-NEXT.

8. Idem para el 10.9.6

9. Idem para el 10.9.16

10. Idem para el 10.9.19

11. Escribir un programa que provoque una impresión tal como:

```
***************
```

usando un ciclo FOR-NEXT que imprima un asterisco en cada pasada.

12. Analizar el siguiente programa y escribir su salida:

```
1Ø INPUT M,N
2Ø FOR I=1 TO M
3Ø PRINT "*";
4Ø NEXT I
5Ø PRINT
6Ø FOR I=1 TO N
7Ø PRINT "$";
8Ø NEXT I
9Ø END
```

13. Idem a 12 para:

```
1Ø INPUT N
2Ø FOR I=1 TO N
3Ø INPUT M
4Ø FOR J=1 TO M
5Ø PRINT "*";
6Ø NEXT J
7Ø PRINT
8Ø NEXT I
9Ø END
```

14. Haciendo uso del ejercicio 13, dibujar el siguiente diagrama

```
****
*********
******
**
*********
```

12

Variables subindicadas

12.1. VARIABLES CON UN SUBINDICE

En Basic pueden usarse variables subindicadas, tales como:

A(1), A(2), A(3), P(7), T(I) o X(I–J–2)

Su forma general es:

variable (índice)

donde variable ha de ser una letra de la A a la Z, y el índice ha de ser una constante, una variable o una expresión aritmética, pero que en cualquier caso su valor ha de ser un número entero positivo.

EJEMPLO 1

Si I = 3 y J = 2, son válidas:

A(7)
B(I)
C(I–J+8)
B(6+2*J–I)

en cambio, no serían válidas:

C(I/2)
A(1.27)
AB(17)

las dos primeras porque los índices no son enteros y la última porque la variable no es monoliteral.

Estas variables son muy útiles cuando se han de manejar grupos de números; supongamos que un grupo de 7 alumnos obtiene las siguientes notas en latín:

6, 5, 7.5, 3, 9, 4, 5

si queremos guardarlas en memoria, debemos asignar una variable a cada alumno, tales como A, B, C, D, E, F, G y después proceder con una instrucción de entrada:

```
3Ø A,B,C,D,E,F,G
```

Si en lugar de 7 alumnos fuesen 40, las dificultades para la asignación de las variables a cada alumno y el recordar después qué variable corresponde a cada alumno puede alcanzar un grado de dificultad bastante alto.

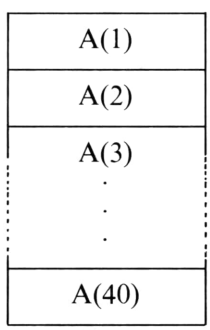
Así pues, lo que vamos a hacer es designar el primer alumno por A(1), el segundo por A(2), y así sucesivamente hasta A(7) ó A (40), según el caso.

Para entrar las notas basta hacer uso de una instrucción FOR-NEXT:

```
2Ø FOR I=1 TO 4Ø
3Ø INPUT A(I)
4Ø NEXT I
```

En esta parte de programa, al llegar a la instrucción 20, I toma el valor 1 y en 30 entramos A(1); volvemos, debido al NEXT I, a la instrucción 2Ø e I toma el valor 2, con lo cual en 30 entramos A(2); así sucesivamente, en cada vuelta entramos la nota de un alumno; el alumno que figure en el noveno lugar de la lista entrará en la novena vuelta y ocupará la posición de memoria A(9).

Estos valores se almacenan en memorias consecutivas, tales como:



Una variable subindicada con un solo subíndice suele llamarse un vector; volviendo al caso de las siete notas anteriores, en lugar de escribir:

A(1)=6 A(2)=5 A(3)=7.5 A(4)=3 A(5)=9 A(6)=4 A(7)=5

para referirnos a ellas manualmente, escribiremos:

```
(6, 5, 7.5, 3, 9, 4, 5)
```

o, más aún, para saber que la variable subindicada usada es la A:

```
A = (6, 5, 7.5, 3, 9, 4, 5)
```

A representa, pues, un vectos de siete componentes designadas por A(1) hasta A(7).

Es importante resaltar que esta notación es de uso externo al ordenador para simplificar escrituras manuales; dentro de un programa solo podemos referirnos, en un momento dado, a una componente A(I) y nunca a un vector completo. Es más, pueden usarse simultáneamente la variable A y la variable subindicada A(I).

EJEMPLO 2

Las instrucciones siguientes son correctas:

```
1Ø INPUT A
2Ø FOR I=1 TO 7
3Ø INPUT A(I)
4Ø NEXT I
```

En 1Ø se entra un valor para la variable A y de 20 a 40 se entran los valores de A(1) hasta A(7).

Antes de trabajar con una variable subindicada hay que advertir al ordenador para que reserve memoria suficiente para almacenar todos los valores; para ello se usa la instrucción DIMENSION o simplemente DIM, cuyo formato es:

etiqueta DIM variable (valor máximo del subíndice)

EJEMPLO 3

```
1Ø DIM A(2Ø)
```

con esta instrucción, el ordenador reserva 20 posiciones de memoria designadas por A(1) hasta A(20); nos indica que el valor máximo que puede tomar el subíndice es 20; si usamos A(21) daría error.

Por otra parte, tampoco quiere decir que las 20 memorias deban ser ocupadas; podemos trabajar con un vector de 5 componentes:

A(1), A(2), A(3), A(4), A(5)

y lo único que ocurre es que las memorias 6 a 20 quedan sin utilizar.

En resumen, esta instrucción indica que podemos trabajar con vectores de hasta 20 componentes.

En algunos ordenadores, el Ø sirve también como subíndice, y para ellos DIM A(2Ø) reservaría 21 posiciones de memoria incluyendo el A(Ø)

EJEMPLO 4

Vamos a entrar N números y conseguir que aparezcan en orden inverso al de entrada; $N \leqslant 100$.

El proceso que seguiremos será:

a) Con DIM reservar 100 posiciones de memoria.
b) Entrar el número de componentes N.

c) Entrar el primer número en B(1), el segundo en B(2), . . . el N-simo en B(N).
d) Mostrar desde B(N) hasta B(1), con un bucle FOR de incremento de la variable −1.

El programa es:

```
1Ø DIM B(1ØØ)
2Ø INPUT "NUMERO DE COMPONENTES=";N
3Ø FOR I=1 TO N
4Ø INPUT B(I)
5Ø NEXT I
55 PRINT "LA ENTRADA HA TERMINADO"
6Ø FOR I=N TO 1 STEP -1
7Ø PRINT "ELEMENTO NUMERO";I;"=";B(I)
8Ø NEXT I
9Ø END
```

En el caso de que N = 5 y los números sean 4, 6, 2, 8, 3 la salida es:

```
ELEMENTO NUMERO 5= 3
ELEMENTO NUMERO 4= 8
ELEMENTO NUMERO 3= 2
ELEMENTO NUMERO 2= 6
ELEMENTO NUMERO 1= 4
```

Las instrucciones 6Ø, 7Ø y 8Ø pueden ser sustituidas por las siguientes con un efecto idéntico.

```
6Ø FOR I=1 TO N
7Ø PRINT "ELEMENTO NUMERO ";N-I+1;" = ";B(N-I+1)
8Ø NEXT I
```

EJEMPLO 5

En el ejemplo 4, en lugar de imprimir los números en orden inverso al de entrada, calculamos la suma de todos los números. Como siempre que hay que calcular una suma de un conjunto de números la inicializamos a cero S=0.

```
1Ø DIM B(1ØØ)
2Ø INPUT "NUMERO DE COMPONENTES=";N
3Ø FOR I=1 TO N
4Ø INPUT B(I)
5Ø NEXT I
6Ø S=Ø
7Ø FOR I=1 TO N
8Ø S=S+B(I)
9Ø NEXT I
1ØØ PRINT "LA SUMA VALE ";S
11Ø END
```

Este programa puede simplificarse si vamos calculando la suma de los números según van etrando:

```
1Ø DIM B(1ØØ)
2Ø INPUT "NUMERO DE COMPONENTES";N
3Ø S=Ø
```

```
4Ø FOR I=1 TO N
5Ø INPUT B(I)
6Ø S=S+B(I)
7Ø NEXT I
8Ø PRINT "LA SUMA VALE ";S
9Ø END
```

EJEMPLO 6

En una clase con N alumnos $\leqslant$ *40 se han hecho dos exámenes de física; queremos almacenar los resultados de uno y otro examen así como la nota media de ambos.*

Las notas del primer examen serán almacenadas en un vector A de N componentes; las del segundo, en un vector B de N componentes, y las notas medias se almacenarán en el vector M, también de N componentes; imprimir las notas medias.

```
1Ø DIM A(4Ø),B(4Ø);M(4Ø)
2Ø INPUT "NUMERO DE ALUMNOS=";N
25 PRINT "ENTRADA DE NOTAS DEL PRIMER EXAMEN"
3Ø FOR J=1 TO N
4Ø INPUT A(J)
5Ø NEXT J
55 PRINT "ENTRADA DE NOTAS DEL SEGUNDO EXAMEN"
6Ø FOR J=1 TO N
7Ø INPUT B(J)
8Ø NEXT J
9Ø PRINT "CALCULO E IMPRESION DE LAS NOTAS MEDIAS"
1ØØ FOR J=1 TO N
11Ø M(J)=(A(J)+B(J))/2
12Ø PRINT "NOTA MEDIA DEL ALUMNO NUMERO ";J;" = ";M(J)
13Ø NEXT J
14Ø END
```

El programa anterior puede reducirse a un único bucle FOR en el que se entre una nota del primer examen y a continuación la nota del segundo examen para el mismo alumno; calculamos la nota media y la imprimimos antes de pasar a un nuevo alumno.

Dejamos al lector esta simplificación.

El siguiente ejemplo puede no resultar tan obvia su comprensión.

EJEMPLO 7

Disponemos de una lista de N alumnos con sus edades correspondientes; se trata de buscar el más joven de la lista dando su posición y su edad; si hay varios con esta edad mínima nos quedaremos con el que figure en primer lugar.

El procedimiento que vamos a seguir consistirá en ir recorriendo la lista, uno a uno, y quedarnos con la posición del menor que hayamos visto hasta el momento.

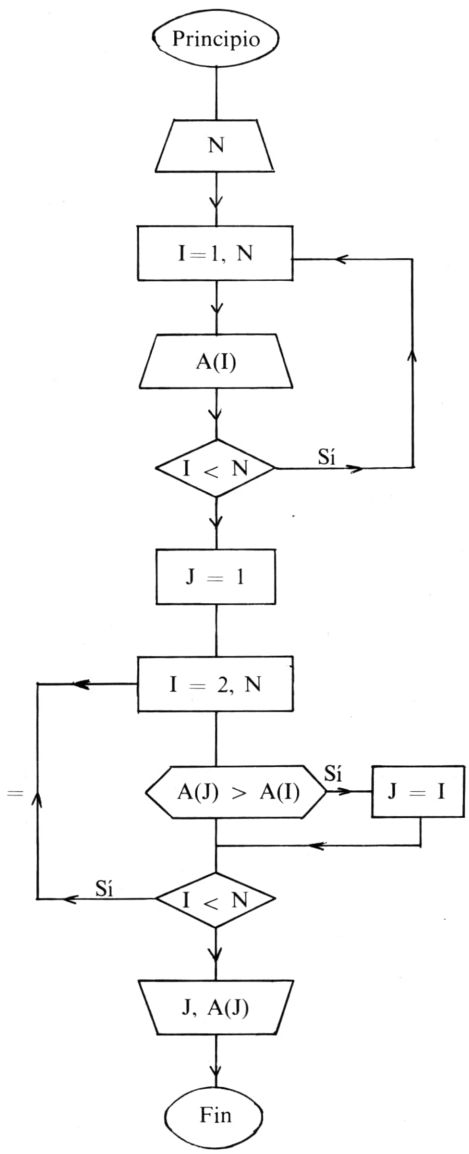
Las edades se almacenarán inicialmente en un vector de N componentes: A(1), A(2), ... A(N), estando en A(1) la edad del primero de la lista, en A(2) la del segundo, ..., etc. Designaremos por J la variable que almacenará la posición de alumno de menor edad de los recorridos hasta el momento;

inicialmente J valdrá 1; la variable I hará de contador desde 2 hasta N y recorrerá los restantes elementos de la lista.

El algoritmo puede ser el siguiente:

a) Dar un valor a N e introducir las N edades.
b) Iniciar J=1 e I=2.
c) Si A(J)>A(I) hacer J=I.
d) Si I<N aumentar I en una unidad y volver a *c*.
e) Mostrar J y A(J).
f) Fin.

En el siguiente organigrama se distinguen claramente cada uno de los anteriores pasos ya desarrollados:



El programa sería, suponiendo N $\leqslant$ 100:

```
1Ø REM BUSCAR EL MAS JOVEN DE UN GRUPO DE ALUMNOS
2Ø DIM A(1ØØ)
3Ø INPUT "NUMERO DE ALUMNOS=";N
4Ø REM ENTRADA DE LAS EDADES
5Ø FOR I=1 TO N
6Ø INPUT A(I)
7Ø NEXT I
8Ø REM COMIENZA LA BUSQUEDA DEL MENOR
9Ø J=1
1ØØ FOR I=2 TO N
11Ø IF A(J)>A(I) THEN J=I
12Ø NEXT I
13Ø PRINT "EL MENOR ES EL ";J;" SU EDAD ES ";A(J)
14Ø END
```

EJEMPLO 8

Ordenar de menor a mayor una lista de números.

Los algoritmos para la ordenación de una lista de números son muy variados; existen los llamados métodos directos, entre los que podemos destacar los siguientes:

a) Ordenación por inserción: directa y binaria.
b) Ordenación por selección directa.
c) Ordenación por intercambio: método de la burbuja y de la sacudida,

Cada uno de ellos tiene diversas variables.

Además de estos métodos directos, algunos muy útiles en listas reducidas, existen otros métodos más sofisticados para la ordenación de listas muy amplias, tales como el método de Shell, el método del montículo y el método rápido, cada uno de los cuales constituye una optimización de los métodos directos *a, b* y *c,* respectivamente.

El *método de inserción* fue usado en 6.2 para ordenar una colección de fichas; a partir de la lista original se va formando una nueva lista, de uno en uno, insertándolo en el lugar que le corresponde; según el camino seguido para hacer la inserción surgen los métodos de inserción directa o binaria.

Como aplicación ordenemos los números 32, 51, 13, 29, 61, 17 y 56 por inserción directa; en cada paso la lista, formada por los siete números, está dividida en dos partes, una de las cuales (la subrayada) contiene los elementos ya ordenados y la otra los elementos que faltan por insertar; inicialmente la parte ordenada sólo contiene el primer elemento.

Inicial	32	51	13	29	61	17	56
Primera inserción	32	51	13	29	61	17	56
Segunda inserción	13	32	51	29	61	17	56
Tercera inserción	13	29	32	51	61	17	56
Cuarta inserción	13	29	32	51	61	17	56
Quinta inserción	13	17	29	32	51	61	56
Sexta inserción	13	17	29	32	51	56	61

El *método de intercambio* consiste en comparar el primer elemento con el segundo y si no están ordenados, intercambiarlos; a continuación repetimos lo anterior con el segundo y, si no están ordenados, intercambiarlos; a continuación repetimos lo anterior con el segundo y tercer elemento y así sucesivamente hasta llegar al final de la lista; en este momento tendremos el mayor elemento situado en la última posición. Haciendo otra pasada de forma análoga, conseguiremos que los dos elementos mayores de la lista estén ordenados al final de la misma. Continuando el procedimiento es seguro que tras N − 1 pasadas tendremos la lista ordenada.

Este caso particular del método de intercambio se conoce como método de la burbuja y lo aplicaremos a la ordenación de los siete números anteriores.

Inicial	32	51	13	29	61	17	56
Primera pasada	32	13	29	51	17	56	61
Segunda pasada	13	29	32	17	51	56	61
Tercera pasada	13	29	17	32	51	56	61
Cuarta pasada	13	17	29	32	51	56	61

En esta ocasión en cuatro pasadas hemos conseguido la ordenación buscada.

El método de selección directa consiste en seleccionar el menor elemento dela lista y colocarlo en primera posición mediante un intercambio; a continuación colocamos en segunda posición el menor de los que restan (excluimos al primero) y así sucesivamente, hasta que nos quede un único número que será el mayor y estará bien ordenado.

Aplicándolo a la misma lista de números anterior, se tiene:

Inicial	32	51	13	29	61	17	56
Primer cambio	13	51	32	29	61	17	56
Segundo	13	17	32	29	61	51	56
Tercero	13	17	29	32	61	51	56
Cuarto	13	17	29	32	61	51	56
Quinto	13	17	29	32	51	61	56
Sexto	13	17	29	32	51	56	61

Las flechas indican los números que han sido intercambiados en cada caso.

Dado que este método utiliza la selección del menor de una lista que ha sido estudiado en el ejemplo 6, lo desarrollaremos en su totalidad, dejando los métodos de inserción e intercambio como ejercicios para el lector interesado.

Así pues, si suponemos que la lista consta de N elementos almacenados en A(1), A(2), ... A(N), el algoritmo de selección directa es el siguiente:

a) Dar un valor a N e introducir los N números.
b) Hacer I = 1.
c) Buscar la posición J del menor elemento desde A(I) hasta A(N).
d) Intercambiar A(J) con A(I).
e) Si queda más de un elemento, aumentar I en una unidad y volver a *c*.
f) Fin.

Siguiendo el algoritmo, observamos que en la primera pasada colocamos el primer elemento, al valor I uno; al volver al paso *c*, I vale dos y en esta segunda pasada colocamos el segundo elemento, etc.

El apartado *c* coincide con el ejemplo 6 con la variante de que el valor incial de J ha de ser I, es decir:

```
 90 J=I
100 FOR K=I+1 TO N
110 IF A(J)>A(K) THEN J=K
120 NEXT K
```

Estas instrucciones seleccionan el menor elemento desde A(I) hasta A(N); haciendo variar la I desde 1 hasta N −1 conseguiremos la ordenación.

El paso *d* necesita usar una variable auxiliar exigida, como sabemos, para cualquier intercambio, y que designaremos por X.

El programa completo sería:

```
 10 REM ORDENACION DE UNA LISTA DE NUMEROS POR SELEC-
    CION DIRECTA
 20 DIM A(100)
 30 INPUT "NUMERO DE ELEMENTOS A ORDENAR=";N
 40 REM ENTRADA DE LOS ELEMENTOS
 50 FOR I=1 TO N
 60 INPUT A(I)
 70 NEXT I
 80 REM COMIENZA LA ORDENACION
 90 FOR I=1 TO N-1
100 REM SELECCION DEL MENOR ELEMENTO DESDE A(I) HASTA
    A(N)
110 J=I
120 FOR K=I+1 TO N
130 IF A(J)>A(K) THEN J=K
140 NEXT K
150 REM INTERCAMBIO DE A(J) CON A(I)
160 X=A(I)
170 A(I)=A(J)
180 A(J)=X
190 NEXT I
200 REM LISTA ORDENADA: COMIENZA LA IMPRESION
210 FOR I=1 TO N
220 PRINT A(I)
230 NEXT I
240 END
```

12.2. VARIABLES CON DOS O MAS SUBINDICES

Consideremos un hotel con 6 plantas y 10 habitaciones por planta; si representamos por 1 la habitación ocupada y por un 0 la habitación libre, el siguiente cuadro puede expresar la ocupación del hotel en un momento determinado:

Número de planta 1	*Número de habitación* 1	2	3	4	5	6	7	8	9	10
1	1	0	1	1	0	1	0	1	0	1
2	1	1	1	1	0	1	1	1	1	0
3	0	0	1	0	1	1	1	0	0	0
4	1	1	0	1	1	0	1	1	0	1
5	0	1	1	0	0	1	0	0	1	0
6	1	0	1	1	1	0	0	0	0	0

Así, la habitación de la cuarta planta, número 5 está ocupada, pues figura un 1 en esa posición; podeos simplificar esto escribiendo que:

```
H(4, 5) = 1
```

el primer número indica la planta y el segundo indica el número de habitación dentro de esa planta. Tenemos también que:

```
H(3, 9) = 0     vacía
H(6, 1) = 1     ocupada
H(2, 7) = 1     ocupada
```

Así pues, una habitación cualquiera tiene la representación H(I, J), donde varía de 1 a 6, y J varía de 1 a 10; en el cuadro I indica la fila y J la columna; H(3, 9) está en la intersección de la fila 3 y la columna 9.

Diremos que H(I, J) es una variable con dos subíndices; estas variables deben ser dimensionadas al principio del programa; la entrada de la tabla anterior podría ser:

```
1Ø DIM H(6,10)
2Ø FOR I=J TO 6
3Ø FOR J=1 TO 1Ø
4Ø INPUT H(I,J)
5Ø NEXT J
6Ø NEXT I
```

Esta entrada, según lo estudiado en los bucles anidados, se haría por filas, pues cuando I = 1, J varía de 1 a 10 y entran las habitaciones de la primera planta; a continuación I = 2 y J vuelve a variar de 1 a 10 entrando así la segunda planta; cuando I = 6 entra la sexta planta y termina la entrada.

Si hubiésemos puesto primero la J y después la I, la entrada se haría por columnas.

La salida del cuadro anterior puede hacerse:

```
1ØØ FOR I=1 TO 6
11Ø FOR J=1 TO 10
12Ø PRINT H(I,J);"  ";
13Ø NEXT J
14Ø PRINT
15Ø NEXT I
```

Esta salida sería por filas tal como está escrito anteriormente; el punto y coma de la instrucción 120 provoca que todas las habitaciones de una planta se escriban en el mismo renglón; sólo cuando hay que cambiar la planta hay que pasar por la instrucción 140 se provoca el salto a la línea siguiente.

A las variables de dos subíndices las llamaremos matrices de dos dimensiones.

EJEMPLO 2

Suponemos que en un programa ya figuran las instrucciones:

```
1Ø DIM A(20,8)
2Ø INPUT N,M
```

siendo $N \leqslant 20$ y $M \leqslant 8$; construyamos las siguientes partes de este programa:

a) Entrar por filas una matriz de M filas y N columnas.
b) Imprimir sus elementos en una sola columna.
c) Imprimir sus elementos por filas.
d) Imprimir sus elementos por columnas.

a)

```
3Ø FOR I=1 TO M
4Ø FOR J=1 TO N
5Ø INPUT A(I,J)
6Ø NEXT J
7Ø NEXT I
```

b)

```
1ØØ FOR I=1 TO M
11Ø FOR J=1 TO N
12Ø PRINT A(I,J)
13Ø NEXT J
14Ø NEXT I
```

c)

```
2ØØ FOR I=1 TO M
21Ø FOR J=1 TO N
22Ø PRINT A(I,J);"  ";
23Ø NEXT J
24Ø NEXT I
```

d)

```
3ØØ FOR J=1 TO N
31Ø FOR I=1 TO M
32Ø PRINT A(I,J);"  ";
33Ø NEXT I
34Ø NEXT J
```

Se pueden definir variables con tres o más subíndices; por ejemplo, supongamos que en un centro hay 10 grupos de alumnos de primer curso que de lunes a viernes dan 7 horas diarias de clase de un conjunto de 10 asignaturas; podemos convenir en la siguiente expresión:

Clase (grupo, día de la semana, hora de ese día) = número de la asignatura que se imparte en esa hora.

Así:

C(2, 4, 6) = 7

se lee; que el segundo grupo, el jueves a sexta hora esta recibiendo la clase correspondiente a la asignatura número siete.

C(5, 1, 3) = 9

el quinto grupo, el lunes a la tercera hora tiene clase de la asignatura incidada con el número 9.

La introduccion del horario de estos diez grupos de primero se puede hacer de la siguiente forma:

```
1Ø DIM C(6,5,7)
2Ø FOR I=1 TO 6
3Ø FOR J=1 TO 5
4Ø FOR K=1 TO 7
5Ø INPUT C(I,J,K)
6Ø NEXT K
7Ø NEXT J
8Ø NEXT I
```

EJERCICIOS

1. La lista siguiente corresponde a las 12 provincias españolas con mayor superficie:

Albacete	14 862	Huesca	15 686
Badajoz	21 659	León	14 072
Burgos	14 328	Sevilla	14 010
Cáceres	19 945	Teruel	14 796
C. Real	19 750	Toledo	15 345
Cuenca	17 056	Zaragoza	17 135

Numéralas e introduce sus superficies en un vector de 12 componentes.

2. La puntuación alcanzada por los ocho primeros equipos en un campeonato es:

34 29 24 23 20 19 19

almacena estos datos en un vector de ocho componentes. Imprímelos seguidamente:

a) Todos sobre una línea.
b) Cuatro por línea.
c) Uno en cada línea.

3. En un centro escolar donde los grupos no sobrepasan los 30 alumnos, queremos hacer un programa que nos sirva para almacenar las notas de un examen de latín, de cualquier grupo, dándole previamente el número de alumnos que lo componen.

4. Ampliar el programa del ejercicio 3 para que una vez introducidas todas las notas calcule la media aritmética de todas ellas.

5. Ampliar el programa del ejercicio 3 para que nos muestre al final el número de aprobados y suspensos.

6. El profesor de latín que dio las notas en el ejercicio 3, observa que hay demasiados suspensos y decide lo siguiente: si el número de aprobados es menos de la mitad, subir a todos los suspensos un punto; si aún continúan aprobando menos de la mitad, volver a subir otro punto; así sucesivamente, hasta conseguir que el número de aprobados supere la mitad de la clase. Desarrolla un programa que consiga este objetivo y mostrar las notas finales junto con el número de la lista correspondiente al alumno cuya nota aparece.

7. Lanzamos dardos sobre una diana y obtenemos las puntuaciones 0, 1, 2 ó 3, según la posición conseguida; si hacemos N lanzamientos, escribir un programa al que podamos ir entrando las puntuaciones de cada tirada y al final nos dé la suma de los puntos obtenidos, así como el número de ceros obtenidos, el de unos, el de dos y el de tres (usar variable subindicada y ningún IF).

8. Repetir el ejercicio 7 suponiendo que las puntuaciones que se pueden conseguir son de cero a diez.

9. Almacenar en un vector de 60 componentes los senos de los ángulos desde un agrado hasta 60 grados. Imprimir estos datos en dos columnas:

ANGULO SENO

10. La diagonal de un ortoedro de aristas a_1, a_2 y a_3 se calcula mediante la fórmula:

$$D = \sqrt{a_1^2 + a_2^2 + a_3^2}$$

Almacena las tres aristas en un vector de tres componentes y hallar luego la diagonal.

11. Dados los vectores $A = (a_1, a_2, \ldots, a_n)$ y $B = (b_1, b_2, \ldots, b_n)$, su producto escalar viene definido por:

$$P = A \cdot B = \sum_{i=1}^{n} a_i \cdot b_i$$

Calcula el producto escalar mediante un programa Basic.

12. las puntuaciones de matemáticas y física de un grupo de 15 alumnos son:

Alumno número	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Matemáticas	5	3	7	4	2	3	7	9	2	4	6	1	6	8	2
Física	6	2	6	3	4	6	5	7	4	6	3	3	5	5	5

Almacenar dichas notas en una matriz, e imprimirla en una tabla con la cabecera:

ALUMNO NUMERO MATEMATICAS FISICA

13. En el ejercicio 3 conseguir una impresión de un diagrama de barras tal como:

```
0  ****
1  **********
2  ***
3  *******
```

correspondiendo el número de asteriscos con el número de veces que hemos obtenido la puntuación previa.

14. Un grupo de gimnastas participan en cinco pruebas distintas, calificadas de 1 a 100; almacenar la puntuación que consiguió cada uno de ellos en cada prueba, en una matriz tal como:

A(I,J)=K

I debe indicar el dorsal del gimnasta
J el número de la prueba
K la puntuación obtenida por el gimnasta I en la prueba J.
Imprimir estos valores sobre una tabla:

Dorsales	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5
1					
2					
3					
...					

15. En el ejercicio 14, una vez introducidos los datos de la matriz, ampliar el programa y sumar las filas para obtener la puntuación total obtenida por cada participante.

16. El jurado del ejercicio 14, adopta el acuerdo de ir restando a los 100 puntos iniciales la cantidad de 5 puntos por cada error que cometa el participante; curiosamente, como algunos eran malísimos, aparecieron puntuaciones negativas para algunos de ellos en ciertas pruebas.

Ampliar el programa del ejercicio 14 para que, una vez introducidos los datos, se haga una revisión de la matriz y se sustituyan las puntuaciones negativas por ceros.

17. Continuando con el ejercicio 14, queremos asignar un grado de dificultad a cada prueba y pensamos que un coeficiente indicativo podría ser:

$$C = \frac{100 \times \text{número de participantes}}{\text{suma de puntuaciones dadas en esa prueba}}$$

de esta forma, el coeficiente será mayor para las pruebas en las que se han dado menores puntuaciones.

A partir de la matriz introducida en el ejercicio 14, hallar este coeficiente de dificultad para cada prueba.

18. Escribir un programa que nos muestre el mayor elemento de una matriz y su posición.

19. Idem para el menor.

20. Una matriz es cuadrada si tiene el mismo número de filas que de columnas. Leer una matriz cuadrada A, y a partir de ella construir una matriz B, tal que:

B(I, J) = 0 si I es distinto de J
B(I, J) = A(I, J) si I es igual a J

21. Escribir un programa Basic que nos diga de cuántas formas puede obtenerse la puntuación 9 al tirar tres dados. Repetir el ejercicio para una puntuación K cualquiera y los resultados, al variar K, almacenarlos en un vector.

22. Construir un programa que archive todos los posibles resultados de una quiniela con cinco partidos. Imprimir únicamente aquellos en los que se dé al menos dos empates.

23. Ordenar de menor a mayor los elementos de cada fila de una matriz.

24. La suma de dos matrices, A y B, que tengan igual número de filas y de columnas se define como otra matriz C, tal que:

C(I, J) = A(I, J) + B(I, J)

es decir, los elementos C son la suma de los elementos de A y de B que ocupan la misma posición.

Construir un programa que sume matrices del mismo orden.

25. En un vector de 100 componentes guardar los 100 primeros números primos e imprimirlos:

a) Uno por fila.
b) Cinco por fila.
c) Siete por fila.

26. Escribir un programa que calcule el factorial de un número N con todas sus cifras. Aplicarlo al cálculo del factorial de 100.

13

Variables alfanuméricas

13.1. CADENAS

Cuando hablamos de un carácter alfanumérico nos referiremos indistintamente a:

— Un dígito, 0, 1, 2, 3, 4, 5, 6, 7, 8 ó 9.
— Una letra de la A a la Z
— Cualquiera de los símbolos de puntuación, separación u operadores: ( ) * + − / = : . , ;
— Un símbolo especial como: ! " ¿ ? & $ % etc.

Llamaremos cadena a toda sucesión de caracteres alfanuméricos entre comillas; estas cadenas han sido ya usadas con las instrucciones INPUT y PRINT; por ejemplo, en:

```
2Ø INPUT "VALOR DE X1=";X
11Ø PRINT "LA VARIABLE X NO ES IGUAL A CERO"
```

Para evitar confusiones no usaremos las comillas como elementos de una cadena.

Lo fundamental de este apartado radica en el hecho de que en Basic puede igualarse una variable a una cadena tal como sigue:

```
6Ø A$="PEPE"
```

o, en general,

el nombre de la variable = cadena

El nombre de la variable ha de ser el de una variable numérica terminada en $; en este momento A$ contiene la cadena PEPE, pero si ponemos a continuación:

```
7Ø A$="FLOR"
```

el contenido de A$ es ahora la cadena FLOR.

Así pues, A$ es una variable cuyo contenido en lugar de ser un número es una cadena, de aquí que la llamemos variable alfanumérica, variable de tipo cadena o cadena variable. Al contenido de A$ es susceptible de ser modificado tal como ocurría con las variables numéricas.

La longitud permitida de las cadenas, es decir, su número de caracteres, depende del modelo de ordenador; el margen de variación es muy amplio, desde 6 ó 7 a varios miles de carácteres. Para nosotros nos bastará con que acepte unos 30 caracteres.

EJEMPLO 1

```
1Ø INPUT B$,K$
2Ø T$="CADA DIA LUCE EL SOL"
3Ø PRINT B$;K$;T$
4Ø END
```

Al pasar este programa y pedirnos el primer dato, B$, pdemos escribir cualquier cadena, tal como:

```
"PEPE PEREZ"
```

pulsamos RETURN y nos pide K$ y escribimos:

```
"MIERCOLES"
```

El resultado del PRINT será:

```
PEPE PEREZ MIERCOLES CADA DIA LUCE EL SOL
```

sin que salgan, como vemos, las comillas en la impresión de resultados.

Las variables de cadena pueden dimensionarse como las numéricas:

```
1Ø DIM A$(15)
```

define un vector con 15 componentes cada una de las cuales es una cadena.

EJEMPLO 2

El siguiente programa almacena N nombres ($N \leqslant 40$) en una variable de cadena dimensionada e imprime la lista al final.

```
1Ø DIM L$(40)
2Ø INPUT N
```

```
30 FOR I=1 TO N
40 INPUT L$(I)
50 NEXT I
60 FOR J=1 TO N
70 PRINT L$
80 NEXT J
```

EJEMPLO 3

En un bloque de pisos de 5 plantas, con tres pisos por planta, representaremos por N$(I, J) *el nombre del propietario del piso situado en la planta I piso J, es decir:*

N$(4,2) se refiere a la planta 4 piso número 2
N$(1,3) se refiere a la planta 1 piso número 3

El siguiente programa introduce los nombres de todos los propietarios y provoca una salida en la forma:

```
EL PROPIETARIO DEL PISO 1 PLANTA 1 ES D. nombre
EL PROPIETARIO DEL PISO 2 PLANTA 1 ES D. nombre
...
EL PROPIETARIO DEL PISO 3 PLANTA 5 ES D. nombre
```

```
10 DIM A$(5,3)
20 FOR I=1 TO 5
30 FOR J=1 TO 3
40 PRINT "PLANTA ";I;" PISO ";J
50 INPUT "NOMBRE=":A$(I,J)
60 NEXT J
70 NEXT I
80 FOR I=1 TO 5
90 FOR J=1 TO 3
100 PRINT "EL PROPIETARIO DEL PISO ";J;
    " PLANTA ";I;" ES D. "; A$(I,J)
110 NEXT J
120 NEXT I
130 END
```

13.2. CODIGO ASCII

La representación interna de los caracteres alfanuméricos, como sabemos de los primeros capítulos ha de ser en binario y, por lo tanto, cada carácter ha de tener un número asociado; es decir, la letra A, por ejemplo, no puede almacenarse en memoria tal cual es, ya que en memoria sólo se admiten sucesiones de unos y ceros; entonces, cuando la letra A aparece formando parte de una cadena se le asigna un valor número para su representación interna.

El valor asignado a cada carácter varía según el código utilizado por el

ordenador; actualmente el más usado es el código ASCII esquematizado en la siguiente tabla:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
16																
32	♭	!	"		$	%	&		(	)	*	+	,	−	.	/
48	Ø	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[		]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z					

El valor asignado a un carácter se obtiene sumando la fila y la columna a la que pertenece. El blanco está indicado por ♭. Los espacios blancos de la tabla corresponden a caracteres de control.

Teniendo en cuenta esta representación numérica interna para las cadenas podemos intercalar un operador de relación entre dos cadenas; así:

A$ < B$

esta desigualdad es cierta si alfabéticamente A$está antes que B$, es decir, si la letra con la que comienza A$ está antes en el alfabeto que la letra con la que comienza B$ o si la primera letra en que difieran ambas cadenas, comenzando por la izquierda, es anterior en el alfabeto de la A$ que la de B$.

Análogamente podemos usar cualquiera de los operadores:

$$= \quad < \quad > \quad \leqslant \quad \geqslant \quad \neq \quad (< >)$$

EJEMPLO 1

Entrar dos nombres y que salgan escritos en orden alfabético.

Este ejemplo es equivalente al de entrar dos números y mostrarlos ordenados de menor a mayor y la única diferencia es la designación de las variables:

```
1Ø INPUT "NOMBRE 1:";A$;"NOMBRE 2:";B$
2Ø IF A$⩽B$ THEN 6Ø
3Ø C$=A$
4Ø A$=B$
5Ø B$=C$
6Ø PRINT A$,B$
7Ø END
```

EJEMPLO 2

Vamos a buscar en una lista de nombres el que iría en primera posición por orden alfabético.

Copiamos el programa del ejemplo 12.1.6 de buscar el menor de una lista de números cambiando la varable A(I) por A$(I):

```
1Ø REM BUSCAR EN UNA LISTA DE NOMBRES EL
   PRIMERO POR ORDEN ALFABETICO
2Ø DIM A$(100)
3Ø INPUT "NUMERO DE NOMBRES DE LA LISTA=";N
4Ø REM ENTRADA DE NOMBRES
5Ø FOR I=1 TO N
6Ø INPUT A$(I)
7Ø NEXT I
8Ø REM COMIENZA LA BUSQUEDA
9Ø J=1
1ØØ FOR I=2 TO N
11Ø IF A$(J)>A$(I) THEN J=I
12Ø NEXT I
13Ø PRINT "EL PRIMERO SERIA ";A$(J)
14Ø END
```

La ordenación de una lista de nombres por orden alfabético se corresponde con el ejemplo 12.1.7 que ordena una lista de números de menor a mayor. El lector interesado puede hacer la sustitución de las variables numéricas por alfanuméricas.

13.3. FUNCIONES PARA CADENAS

Existen un grupo de funciones especiales aplicables a cadenas y variables de cadena, que estudiamos a continuación.

1. ASC(A$S)

Da el código ASCII del primer carácter de A$
Así, se tiene:

```
ASC("A")    da 65
ASC("ARUSA")    da también 65
```

Si B$="N" y C$="FLECHA")

```
ASC(B$)    da 78      y
ASC(C$)    da 70
```

2. CHR$(X)

Si X es un número del código ASCII, esta función devuelve el carácter correspondiente a ese código.

```
4Ø PRINT CHR$(65)      imprime la letra A
5Ø PRINT CHR$(78)      imprime la letra N
6Ø PRINT CHR$(70)      imprime la letra F
```

Esta función es la inversa de la anterior y podemos escribir:

```
7Ø PRINT CHR$(ASC("LIBRO"))
```

Como ASC("LIBRO") = 76 que el código dela letra L, la instrucción 70 imprimiría una L.

```
8Ø X=ASC("COPY")
9Ø PRINT CHR$(X)
```

la instrucción 90 imprimiría la letra C; el valor de X es 67.

3. LEFT$(A$,P)

Esta función origina otra cadena compuesta por los P caracteres de A$ situados más a la izquierda:

```
1Ø A$="IMPARES"
2Ø B$=LEFT$(A$,2)
3Ø C$=LEFT$(A$,3)
4Ø D$=LEFT$(A$,4)
5Ø E$=LEFT$(A$,5)
```

Los respectivos valores de B$=LEFT$(A$,, C$=LEFT$(A$,, D$=-LEFT$(A$, y E$=LEFT$(A$, son:

```
IM
IMP
IMPA
IMPAR
```

4. RIGHTS$A($,Q)

Esta función es análoga a la anterior pero considera los Q caracteres de A$ situados más a la derecha. Así:

```
1Ø A$-"IMPARES"
2Ø FOR I=1 TO 7
3Ø B$=RIGHT$(A$,I)
4Ø PRINT B$
5Ø NEXT I
6Ø END
```

La impresión sería:

```
S
ES
RES
ARES
PARES
MPARES
IMPARES
```

5. MID$(A$,P,Q)

Esta función origina una cadena compuesta por los Q caracteres de A$ a partir del carácter P (incluido éste):

```
1Ø Ø F$="CONTROLADOR"
11Ø T$=MID$(F$,2,3)
12Ø R$=MID$(F$,3,5)
13Ø H$=MID$(F$,4,2)
```

Los respectivos valores de T$, R$ y H$ son:

```
ONT
NTROL
TR
```

6. LEN(A$)

Nos da la longitud de la cadena A$ incluido los blancos. LEN("CJBFHRB") nos daría el número 7 y LEN("LIBRO NUEVO") nos daría el número 11.

Los siguientes ejemplos nos proporcionan aplicacones interesantes de estas funciones:

EJEMPLO 1

Se trata de contar el número de apariciones de la letra A en la cadena A$ que entramos:

```
1Ø INPUT A$
2Ø T=LEN(A$)
3Ø C=Ø
4Ø FOR I=1 TO T
5Ø IF MID$(A$,I,1)="A" THEN C=C+1
6Ø NEXT I
7Ø PRINT C
8Ø END
```

El procedimiento seguido consiste en:

— Entrar la cadena.
— Hallar su longitud.
— Comparar sus caracteres uno a uno con "A" y si coincide aumentar C en una unidad hasta terminar la cadena.

Observar que la función MID$(A$,I,1) es el carácter I de la cadena A$.

EJEMPLO 2

Vamos a contar el número de palabras que componen una cadena.

El proceso es idéntico al ejemplo 1; cada vez que nos encontremos con un blanco aumentaremos el contador de palabras en una unidad. El programa es el mismo sustituyendo en la instrucción 50 "A" por " " o por:

```
ASC(MID$(A$I,1))=32
```

ya que:

ASC("␣")=32

EJEMPLO 3

Supongamos que N$ contiene el nombre y dos apellidos de una persona (todos en una palabra); se trata de crear dos nuevas variables M$ y P$ que contengan la primera sólo el nombre y la segunda los dos apellidos.

Buscaremos como antes el primer blanco que apareza y consideraremos como M$ la palabra a la izquerda del blanco (nombre) y como P$ lo que hay a su derecha (dos apellidos).

El programa es el siguiente:

```
1Ø INPUT N$
2Ø T=LEN(N$)
3Ø FOR I=1 TO T
4Ø IF MID$(N$,I,1)=" " THEN 6Ø
5Ø NEXT I
6Ø M$=LEFT$(N$,I)
7Ø P$=IGHT$(N$,T–I)
8Ø PRINT M$,P$
9Ø END
```

En lugar de 40 puede escribirse:

```
4Ø IF ASC(MID$(N$,I,1))=32 THEN 6Ø
```

EJEMPLO 4

Tenemos una lista de N alumnos ($N \leqslant 40$) en la que figuran escritos en primer lugar los apellidos y a continuación el nombre. Se trata de contar cuántos de esos alumnos tienen un apellido determinado como primer apellido.

El algoritmo que seguiremos será el siguiente:

a) Dimensionar una variable A$(40) donde introducir los nombres.
b) Entrar el número N de alumnos.
c) Entrar los nombres.
d) Entrar en B$ el apellido a buscar; calcular su longitud L.
e) Iniciar una variable contador a cero.
f) Leer los primeros L caracteres de cada nombre y, si coinciden con B$, sumar uno a la variable contador.
g) Imprimir el valor del contador.
h) Fin.

El programa sería:

```
1Ø DIM A$(4Ø)
2Ø INPUT N
3Ø FOR I=1 TO N
4Ø INPUT A$(I)
5Ø NEXT I
```

```
 6Ø INPUT B$
 7Ø L=LEN(B$)
 8Ø C=Ø
 9Ø FOR I=1 TO N
1ØØ IF LEFT$(A$(I),L)=B$ THEN C=C+1
11Ø NEXT I
12Ø PRINT C
13Ø END
```

EJEMPLO 5

Vamos a introducir un grupo de palabras en el ordenador y queremos obtener, al finalizar la entrada, un listado de las distintas palabras introducidas y el número de veces que se han repetido cada una de ellas.

Consideraremos que el texto a introducir tiene menos de 1000 palabras distintas que serán introducidas en la variable A$(I),I=1,1000 según vayan apareciendo; la variable que servirá de contador de las apariciones de A$(I) será A(I).

El proceso lo resumimos como sigue:

a) Dimensionar AS(1000) y A(1000).
b) Inicializar todas las variables A(I) a cero.
c) Entrar la primera palabra A$(1) y aumentar A(1) en una unidad.
d) Entrar la palabra siguiente o una @ si se ha terminado.
e) Si ha aparecido antes, aumentar el contador de esta palabra en una unidad. Si no apareció antes ocupar la primera posición libre A$(I) y hacer A(I)=1. Si se introdujo el carácter *a* ir a *g*.
f) Volver a *d*.
g) Fin.

El programa es:

```
 1Ø DIM A$(1000),A(1000)
 15 REM SE INICIALIZAN LAS VARIABLES A(I) A CERO
 2Ø FOR I=1 TO 1000
 3Ø A(I)=Ø
 4Ø NEXT I
 45 REM COMIENZA LA ENTRADA DE PALABRAS
 5Ø PRINT "AL TERMINAR EL TEXTO ENTRAR EL CARACTER"
 6Ø I=Ø
 7Ø INPUT A$(I)
 8Ø IF A$(I)="@" THEN 17Ø
 85 IF I=1 THEN 15Ø
 9Ø FOR J=1 TO I-1
1ØØ IF A$(I)<>A$(J) THEN 13Ø
11Ø A(J)=A(J)+1
12Ø GO TO 7Ø
13Ø NEXT J
14Ø A(I)=1
15Ø I=I+1
16Ø GO TO 7Ø
```

```
17Ø FOR J=1 TO I-1
18Ø PRINT A$(J);" HA APARECIDO ";A(J);" VECES "
19Ø NEXT J
2ØØ END
```

EJERCICIOS

1. Escribir un programa Basic que dos pregunte nuestro nombre, el número de carnet de identidad, dirección y estado civil. Imprimir todos estos datos.

2. Definir un vector o matriz que almacene los datos del ejercicio 1, correspondientes a un grupo de N personas.

3. Almacenar en un vector de cadenas las provincias del ejercicio 12.1 y en otro vector su superficie en Km^2 e imprimir en dos columnas:

PROVINCIA SUPERFICIE

4. Introducir en un vector los siete días de la semana.

5. Idear algún procedimiento para que suministrándole a un programa la fecha en la que nos encontramos, nos diga la estación a la que pertenece. (Usar, por ejemplo, notación decimal para la fecha.)

6. Escribe un programa Basic en el que entrando un verbo, nos muestre su raíz y su terminación.

7. Conseguir la conjugación del presente de indicativo de cualquier verbo de la primera conjugación y almacenarla en un vector de 6 componentes. Imprimirlo simultáneamnte con otro vector que contenga los pronombres.

8. Teniendo en cuenta la equivalencia entre las notas literales y numéricas establecidas en el ejercicio 10.9.10, almacenar en un vector todas las notas literales y hacer el cambio de notas literales a numéricas, tomando como valor numérico el punto medio del intervalo que representa; por ejemplo, si entramos SUFICIENTE que aparezca la nota 5.5.

9. En el mercado de divisas la cotización de la peseta respecto de las otras cinco monedas indicadas es:

1 dólar USA	125 ptas.
1 franco fracés	18 ptas.
1 libra esterlina	202 ptas.
1 marco alemán	53 ptas.
1 corona sueca	17 ptas.

Almacenar en un vector los seis nombres de las monedas (incluida la peseta) y en otro sus cotizaciones y hacer un programa que nos pregunte la moneda de que disponemos, su cantidad y a qué moneda queremos cambiar y nos aparezca la cantidad equivalente en la nueva moneda.

10. Repetir el ejemplo 4 del tema, pero en esta ocasión contando el número de apariciones del apellido TORRES, tanto en primero como en segundo lugar.

11. Suponiendo que cada variable de cadena admite hasta 80 caracteres, idear algún procedimiento para introducir una página de una novela y contar las apariciones de una palabra dada.

12. Repetir el ejercicio 11, pero obteniendo una lista de los renglones en los que dicha palabra aparece.

13. En una clase con 10 alumnos que tienen cuatro horas de clase diarias, se anota los alumnos que faltan a cada una de ellas. idear algún procedimiento que almacene nombres y faltas y provoque una impresión según el formato:

```
        Alumno                          Hora 1   Hora 2   Hora 3   Hora 4
Nombre 1
Nombre 2
...
Nombre 10
```

Los nombres ocuparán las 22 primeras posiciones; si un alumno falta figurará en el lugar correspondiente la palabra FALTA, y si no debe figurar -----.

14. Un grupo de alumnos se examinan de matemáticas, física, latín y dibujo; queremos introducir las notas usando únicamente las claves SUF (suficiente), INS (insuficiente) y NOT (notable) así como provocar una impresión semejante a la del ejercicio 13. Escribir el programa correspondiente.

CUARTA PARTE

OTRAS INSTRUCCIONES

14

Funciones

14.1. FUNCIONES DEFINIDAS POR EL SISTEMA OPERATIVO

En el capítulo 6 dimos una relación de las funciones matemáticas definidas en la mayoría de los modelos de ordenadores; éstas eran:

SIN(X)	LOG(X)
COS(X)	EXP(X)
TAN(X)	INT(X)
ASN(X)	ABS(X)
ACS(X)	SGN(X)
ATN(X)	SQR(X)
LN(X)	

Más adelante, en el capítulo 13, estudiamos un grupo de funciones aplicables a cadenas; eran:

ASC(A$)	RIGHT$(A$,Q)
CHR$(X)	MID$(A$,P,Q)
LEFT$(A$,P)	LEN(A$)

Además de estas funciones, casi todos los sistemas disponen de una función generadora de números aleatorios; se trata de la función:

RND(X)

Esta funcion genera un número z al azar entre 0 y 1. $0 < z < 1$; el argumento X debe ser un número positivo y es un parámetro que interviene para que se generen en cada caso series de números aleatorios distintas.

EJEMPLO 1.

El siguiente programa genera N números al azar.

```
1Ø INPUT N
2Ø FOR I=1 TO N
```

```
3Ø Z=RND(3)
4Ø PRINT Z
5Ø NEXT Z
6Ø END
```

Así si N = 10, una posible generación sería:

Ø.281	Ø.Ø66
Ø.769	Ø.457
Ø.639	Ø.222
Ø.225	Ø.599
Ø.398	Ø.278

En este programa podemos sustituir el número 3 de la RND(3) por cualquier otro número positivo.

En algunos modelos de ordenadores el argumento X es totalmente ficticio pudiéndose usar el mismo número en todos los programas sin que altere la aleatoriedad de los resultados; en otros casos, como dijimos antes, el argumento interviene decisivamente en la generación de los números y nos pueden aparecer problemas si no conocemos a fondo el funcionamiento de la RND en cada caso particular.

Obsérvese que, aunque los números generados por la función RND(X) están comprendidos entre el 0 y el 1, basta multiplicar la función por un número P y la nueva función P*RND(X) generará números entre 0 y P:

```
18Ø Z=6*RND(2)
```

genera números al azar entre 0 y 6.

EJEMPLO 2

Vamos a simular el lanzamiento de un dado mediante un programa Bsic; es decir, cuando lanzamos un dado nos sale uno de los números 1, 2, 3, 4, 5 ó 6 al azar, y vamos a conseguir que esta decisión aleatoria la tome el ordenador usando la función RND.

Según hemos dicho, los números generados por 6*RND(2) están comprendidos entre 0 y 6, luego la parte entera de estos números será 0, 1, 2, 3, 4 ó 5.

La siguiente tabla muestra 10 números generados por la función RND(X), el valor de 6*RND(X) y la parte entera de este último valor:

RND(2)	6*RND(2)	INT(6*RND(2))
Ø.453	2.718	2
Ø.653	3.918	3
Ø.Ø69	Ø.414	Ø
Ø.633	3.798	3
Ø.683	4.Ø98	4
Ø.264	1.584	1
Ø.Ø14	Ø.Ø84	Ø
Ø.074	5.844	5
Ø.Ø5Ø	Ø.3	Ø
Ø.767	4.6Ø2	4

Si sumamos 1 a esta función se obtiene 1+INT(6*RND(2)), cuyos valores son 1, 2, 3, 4, 5 ó 6; luego, esta función simula el lanzamiento de un dado. El siguiente programa simula N lanzamientos de un dado:

```
1Ø INPUT N
2Ø FOR I=1 TO N
3Ø Y=1+INT(6*RND(2))
4Ø PRINT Y
5Ø NEXT I
6Ø END
```

Los programas que, tal como el anterior, sustituyen la ejecución de un fenómeno de azar real, por un modelo de ordenador, se llaman programas de simulación.

EJEMPLO 3

Se trata ahora de simular el lanzamiento de una moneda.

Teniendo en cuenta que los resultados posibles aquí son únicamente dos, consideramos la función: 2*RND(1) que genera números comprendidos entre 0 y 2 y, por lo tanto:

```
INT (2*RND(1))
```

toma los valores 0 ó 1 al azar; convenimos en que el cero represente cara y el 1 cruz.

El siguiente programa simula el lanzamiento de una moneda, contando el número de caras y el número de cruces que aparecen.

```
1Ø INPUT "NUMERO DE LANZAMIENTOS=";N
2Ø REM C REPRESENTARA CARA Y Z CRUZ
3Ø C=Ø
4Ø FOR I=1 TO N
5Ø Y=INT(2*RND(1))
6Ø IF Y=Ø THEN C=C+1
7Ø NEXT I
8Ø Z=N-C
9Ø PRINT "NUMERO DE CARAS=";C
1ØØ PRINT "NUMERO DE CRUCES=";Z
11Ø END
```

EJEMPLO 4

El ordenador enseña la tabla de multiplicar.

Para ello generaremos dos números aleatorios del 0 al 9 ambos incluidos y el ordenador pedirá el resultado del producto; si se introduce bien felicitará y hará un nueva pregunta y, si no, debe indicar que se ha cometido un error y volverá a la misma pregunta. Si se desea terminar, introducir un número negativo cualquiera.

El programa sería el siguiente:

```
1Ø REM TABLA DE MULTIPLICAR
2Ø X=INT(1Ø*RND(1))
3Ø Y=INT(1Ø*RND(1))
4Ø Z=X*Y
5Ø PRINT "ESCRIBIR EL RESULTADO DE ";X;"*";Y
6Ø PRINT "SI DESEA TERMINAR ENTRAR UN NUMERO NEGATI-
   VO"
7Ø INPUT R
8Ø IF R<Ø THEN 14Ø
9Ø IF Z=R THEN 12Ø
1ØØ PRINT "HA COMETIDO ERROR; VUELVA A INTENTARLO"
11Ø GO TO 5Ø
12Ø PRINT "RESULTADO CORRECTO; LE FELICITO"
13Ø GO TO 2Ø
14Ø END
```

14.2. FUNCIONES DEFINIDAS POR EL USUARIO

El lenguaje Basic dispone de un método para que el usuario pueda incorporar funciones definidas por él mismo, para la realización de cálculos específicos.

Estas funciones se definen según el siguiente formato:
n DEF FN*x*(p_1, p_2, ... p_t) = expresión aritmética o lógica, siendo *n* un número de etiqueta, *x* una letra cualquiera, y p_1, p_2, p_3, ... p_t son parámetros cuyo uso ahora veremos.

EJEMPLO 1

```
1Ø DEF FNA(X,Y)=X/2+Y/3
```

en esta instrucción se define la función FNA(X,Y), cuyo uso a partir de ahora es totalmente equivalente al de la función SIN(X) u otra definida por el sistema.

Así, podemos continuar un programa en la forma:

```
2Ø INPUT A,B,C,D
3Ø Z=FNA(A,B)
4Ø L=FNA(C,D)-3*FNA(5,A)
5Ø PRINT Z,L
6Ø END
```

El valor calculado para Z es A/2+B/3 y el calculado para L es C/2+D/3−3*(5/2+A/3).

Es decir, X e Y son parámetros, que en cada uso de la función reciben valores actuales para unas variables que no tienen por qué ser X e Y, como hemos visto.

EJEMPLO 2

*La siguiente función calcula el valor numérico del polinomio X**3−5*X−9, y se aplica para los valores enteros entre 1 y 100.*

```
1Ø DEF FNH(X)=X**3-5*X-9
2Ø FOR I=1 TO 1ØØ
3Ø E=FNH(I)
4Ø PRINT E,
5Ø NEXT I
6Ø END
```

en salida aparecerían 5 valores por línea en 20 líneas.

Pueden definirse funciones de más de dos parámetros; las siguientes funciones están bien definidas:

```
1ØØ DEF FNP(X,Y,Z)=SQR(X**2+Y**2+Z**2)
11Ø DEF FNK(U,V,W,T)=SIN(U-V)+COS(W+T)
```

EJERCICIOS

1. Simular el lanzamiento de dos monedas de modo que los resultados se obtengan en la forma C-C C-Z Z-C o Z-Z.

2. Lanzamos dos dados y nos interesa la suma de los resultados de ambos. Simular este experimento.

3. Comprobar que en la simulación del lanzamiento de una moneda, los valores de:

$$\text{Frecuencia de caras} = \frac{\text{número de caras}}{\text{número de lanzamientos}}$$

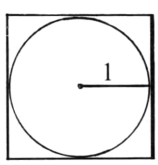
$$\text{Frecuencia de cruces} = \frac{\text{número de cruces}}{\text{número de lanzamientos}}$$

se acercan a las respectivas probabilidades de cara y cruz (0,5) cuando el número de lanzamientos se hace muy grande.

4. idem para un dado; comprobar que las frecuencas de cada cara se acercan a sus correspondientes probabilidades, es decir, a 1/6.

5. Generar un número al azar entre dos números dados P y Q.

6. El área del círculo de la figura es π u.d.s. y la del cuadrado 4 u.d.s. Si lanzamos un dardo y da en el interior del cuadrado, la probabilidad de que dé en el cículo es $\pi/4 \simeq 0{,}785$, es decir:



$$0{,}705 \simeq \frac{\text{área favorable}}{\text{área posible}}$$

Esto significa que de cada N lanzamientos al cuadrado, 0,785 × N deberían dar en el círculo.

Invirtiendo el razonamiento: haciendo 1000 lanzamientos dentro del cuadrado y dividiendo el número de los que han caído en el interior del círculo entre 1000, nos debe dar aproximadamente $0{,}785 = \pi/4$, lo que nos puede servir para hallar el valor de π de forma aproximada:

a) Calcular el valor de π simulando 1000 lanzamientos.
b) Considerar el número $Z = 0{,}4563214$ e $y = 217$ y la sucesión formada como sigue:

$Z_1 = 0{,}4563214$
$Z_2 =$ parte decimal de $217 \times Z_1$
$Z_3 =$ parte decimal de $217 \times Z_2$
. . .
$Z_i =$ parte decimal de $217 \times Z_{i-1}$

se obtiene, así, una sucesión de números que podrían ser considerados como generados al azar.

Simular el lanzamiento de una moneda usando como generador de números aleatorios la anterior sucesión.

8. Comprobar, en el ejercicio anterior, para 1000 y 10 000 lanzamientos, las frecuencias de caras y de cruces. Cambiar los valores de Z e Y y repetir el ejercicio.

9. Definir funciones que provoquen los siguientes efectos:

a) $\dfrac{7{,}8 + x}{x^2 - 7x - 1}$

b) $\dfrac{23 + y}{x - xy + 18}$

c) $\dfrac{1}{\operatorname{sen}^2 x + \sqrt{1 + z^2}}$

d) $2 \log (x + y^2 + 1/x)$

e) $\sqrt{|x^2 + \cos y|}$

10. Definir una función que calcule:

$$3{,}7\sqrt{\left(\frac{x^2}{x + 1/x}\right)}$$

y usarla para calcular:

a) $S = A + 3.7B\sqrt{\left(\dfrac{c^2}{c + 1/c}\right)}$

b) $T = \cos X + 3{,}7\sqrt{\left(\dfrac{(p + 1)^2}{p + 1 + 1/(p + 1)}\right)}$

c) $R = \operatorname{sen} Q + 3{,}7\sqrt{\left(\dfrac{(p - q)^2}{p - q + 1/(p - q)}\right)}$

d) $U = p + 3{,}7\sqrt{\left(\dfrac{B^2(I)}{B(I) + 1/B(I)}\right)}$

11. Definir una función que trunque y otra que redondee a dos decimales, cualquier número decimal, y aplicarlas al cálculo de las raíces cuadradas de los números enteros x tales que $2 \leqslant x \leqslant 100$.

12. Idem para N decimales siendo $0 \leqslant N \leqslant 8$ debiendo figurar N como parámetro.

13. Haciendo uso de la función del ejercicio anterior y suponiendo que en una memoria caben números enteros de 10 cifras, leer individualmente cada una de estas 10 cifras e introducirlas en un vector de 10 componentes.

15

Subrutinas

15.1. SUBRUTINAS

Hemos visto que el uso de las funciones definidas es muy conveniente cuando a lo largo de un programa tenemos que efectuar un determinado cálculo repetidas veces; no obstante, una función debe definirse en una sola instrucción y pudiera ser que en un programa nos interese repetir con frecuencia, no una instrucción, sino un grupo de instrucciones.

Una subrutina es un subprograma del programa principal desde donde puede ser llamada cada vez que se necesite. las subrutinas se colocan al final del programa principal antes del END. Para llamarla desde el programa principal se usa la instrucción:

m GOSUB *n*

m es la etiqueta del GOSUB dentro del programa principal

Con esta instrucción, el control pasa a la línea cuyo número de etiqueta es *n* (donde comienza la subrutina o grupo de instrucciones a ejecutar) y la ejecuta ella, y todas las que le siguen hasta que se encuentra con la instrucción RETURN que provoca la devolución del control al origen de la llamada, es decir, la instrucción siguiente del GOSUB.

Un esquema puede ser el siguiente:

```
1Ø
. .
. .
6Ø GOSUB 2ØØ
7Ø
. .
. .
19Ø GO TO 3ØØ
2ØØ
. .
. .              subrutina
29Ø RETURN
3ØØ END
```

La ejecución de este programa comienza por la instrucción 10 y siguientes; al llegar a la instrucción 60, pasa el control a la línea 200 ejecutándose hasta la

290 en donde al encontrarse con el RETURN salta nuevamente a la instrucción 70, continuando hasta la 190 que envía al EMD terminándose así el programa.

El GOSUB 200 equivale a intercalar en ese punto las instrucciones que definen la subrutina.

EJEMPLO 1

Vamos a calcular $\binom{P}{Q}$ *valiéndonos de una subrutina que calcule el factorial de un número.*

La subrutina debe ser:

```
200 REM SUBRUTINA QUE CALCULA FACTORIAL DE N
210 F=1
215 IF N=0 THEN 250
220 FOR I=1 TO N
230 F=F*I
240 NEXT I
250 RETURN
```

F contiene el factorial de N; N no es aquí un parámetro formal, sino que debe ser una variable actual conteniendo un valor entero positivo.

Para calcular:

$$\binom{P}{Q} = \frac{P!}{Q!(P-Q)!}$$

se puede escribir el siguiente programa:

```
 10 INPUT P,Q
 20 N=P
 30 GOSUB 200
 40 X=F
 50 N=Q
 60 GOSUB 200
 70 Y=F
 80 N=P-Q
 90 GOSUB 200
100 C=X/(Y*F)
110 PRINT P;" SOBRE ";Q;" = ";C
120 GO TO 260
200 REM SUBRUTINA QUE CALCULA EL FACTORIAL DE N
210 F=1
215 IF N=0 THEN 250
220 FOR I=1 TO N
230 F=F*I
240 NEXT I
250 RETURN
260 END
```

Según hemos visto, cada vez que escribimos GOSUB 200 se calcula el factorial del valor que contenga la variable N; de aquí que antes de la llamada

coloquemos en N el valor cuyo factorial deseamos calcular y a la vuelta guardemos en cada caso el valor de F antes de hacer una nueva llamada.

Una subrutina puede llamar a otra subrutina como vemos en el ejemplo siguiente:

EJEMPLO 2

Entrar un número X; una subrutina calculará el valor del polinomio $Y = 2X^2 - 3X + 1$ *y si el valor de Y obtenido es menor que 25 que se llame desde esta subrutina a otra que imprima "EL VALOR DE Y ES MENOR QUE 25".*

Se tiene el siguiente programa:

```
1Ø INPUT X
2Ø GOSUB 3ØØ
3Ø PRINT Y
4Ø GO TO 5ØØ
3ØØ Y=2*X**2-3*X+1
31Ø IF I⩾25 THEN 33Ø
32Ø GOSUB 4ØØ
33Ø RETURN
4ØØ PRINT "EL VALOR DE Y ES MENOR QUE 25"
41Ø RETURN
5ØØ END
```

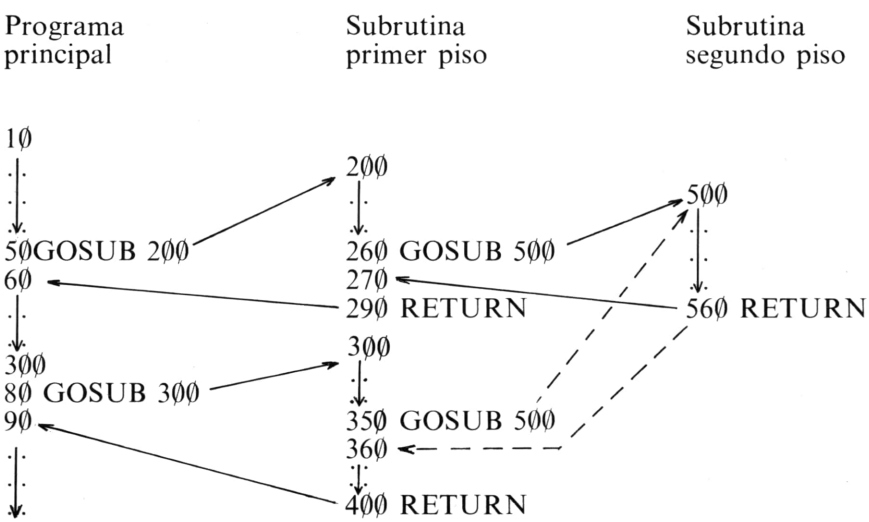
Si X = 1 Y valdría cero y la sucesión de instrucciones a ejecutar sería:

10-10-300-310-320-400-410-330-30-40-500

Si X = 10 Y valdría 171 y la sucesión de instrucciones a ejecutar sería:

10-20-300-310-330-30-40-500

De hecho, pueden formarse sucesivos pisos de llamadas según el siguiente esquema:



Las flechas indican el orden de ejecución; el núero de pisos admisibles varía con el modelo de ordenador.

Podemos establecer ya las diferencias fundamentales entre una función y una subrutina; estas son:

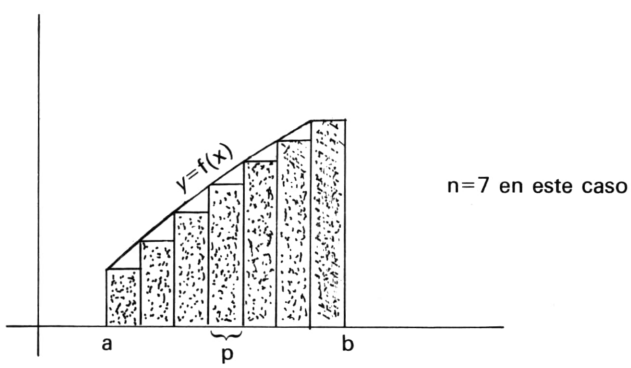
a) La definición de una función ocupa una instrucción y la subrutina puede ocupar varias.

b) En la función aparecen parámetros formales mientras que en la subrutina los valores que se consideran son los reales del programa.

c) El valor devuelto por la función es único, mientras que la subrutina devuelve todos los valores que hayan sido asignados a variables durante su ejecución.

EJEMPLO 3

Consideremos una función y = f(x) *en el intervalo* [a,b]. *Dividimos dicho intervalo en n partes iguales y hallamos la suma de las áreas de los rectángulos inferiores que se forman, según se indica en la figura:*



La base de cada rectángulo es $(b - a)/n = p$ y la altura es el valor de la función en el vértice de la izquierda; los puntos de división son:

$$a, a + p, a + 2p, \ldots a + np$$

el rectángulo I-ésimo tiene como vértices de la base los puntos de abscisas:

$$a + (I - 1)p \qquad \text{y} \qquad a + Ip$$

la altura es $f[a + (I - 1)p]$; esta altura será calculada en una subrutina donde se deberá escribir la función en cada caso particular. Llamaremos S al área pedida.

```
1Ø INPUT "A=";A,"B=";B,"NUMERO DE PARTICIONES=";N
2Ø S=Ø
3Ø P=(B-A)/N
4Ø FOR I=1 TO N
5Ø X=A+(I-1)*P
```

```
6Ø GOSUB 2ØØ
7Ø S=S+P*Y
8Ø NEXT I
9Ø PRINT "LA SUMA DE LAS AREAS DE LOS RECTANGULOS ES
   ":S
1ØØ GO TO 22Ø
2ØØ Y=F(X)   (a escribir la función que proceda)
21Ø RETURN
22Ø END
```

EJERCICIOS

1. Repetir el ejemplo 3 para las áreas de los rectángulos superiores, y comprobar que para funciones continuas la suma de las áreas de los rectángulos inferiores y la de los superiores se aproximan.

2. Leer un vector con 25 componentes; construir una subrutina que calcule la media aritmética, otra el número de componentes positivas y otra la suma de los cuadrados de las componentes.

3. Leer un valor para x y que calcule en una subrutina $y = 3x^3 - 2x^2 + 5x - 2$; si $y \leqslant 24$ que desde esta subrutina llame a otra y calcule $z = y^2 - 7$. Completar con impresiones aclaratorias en cada caso.

4. Dadas tres funciones, f, g y h, hallar $f(g(h(x)))$ mediante tres subrutinas encadenadas.

5. Contruir un programa con cinco niveles de subrutinas de modo que en el programa principal se lea un número N y el efecto de cada subrutina sea el siguiente:

1.ª X=N/2 y si X≤1 llame a subrutina 2.ª
2.ª Y=X/3 y si Y≤1 llame a subrutina 3.ª
3.ª Z=Y/4 y si Z≤1 llame a subrutina 4.ª
4.ª T=Z/5 y si T≤1 llame a subrutina 5.ª
5.ª Imprimir "SE HAN RECORRIDO LOS CINCO NIVELES"

6. De forma semejante a la instrucción ON n GO TO $n_1, n_2, \ldots n_k$ existe la instrucción:

ne ON n GOSUB $n_1, n_2, \ldots n_k$

donde *ne* es un número de etiqueta y n una variable que puede tomar los valores 1, 2, 3, ... k; si n toma el valor i, se ejecutará la subrutina que comienza con la instrucción cuyo número de etiqueta es n_i.

Definir siete subrutinas cada una para imprimir un día de la semana y escribir el programa para que entrando un número del 1 al siete vaya a la subrutina correspondiente e imprima el día de la semana.

16

Las instrucciones READ, DATA y RESTORE

16.1. LAS INSTRUCCIONES READ Y DATA

Hasta ahora, cuando un programa ha necesitado recibir datos para su ejecución hemos usado la instrucción INPUT; no obstante, en ciertos programas es conveniente usar las instrucciones READ y DATA para introducir datos y absolutamente necesario si por alguna razón el programa no va a ser ejecutado inmediatamente después de ser introducido, sino que ha de someterse a una cola de espera que hace que no pueda ejecutarse interactivamente.

La sentencia READ especifica las variables a las cuales serán asignados los valores leídos; esta proposición tiene el siguiente formato:

n READ lista de variables separadas por comas

siendo *n* el número de etiqueta de la instrucción, y las variables pueden ir con o sin subíndices, para valores numéricos o de cadenas.

Son instrucciones READ válidas:

```
 25 READ A,B,C
 4Ø READ A(1),A(7),K
1ØØ READ P,Q,B$,C$(5)
```

los valores para estas variables han de ser leídos de una instrucción DATA, cuyo formato es:

m DATA lista de constantes separadas por comas

siendo *m* el número de etiqueta de la instrucción y las constantes pueden ser numéricas o cadenas de caracteres.

Ejemplos de DATA son:

```
 6Ø DATA 7,5,–2,1.9,18.43,5
11Ø DATA 6,"PUNTO",8
14Ø DATA "LILA","FLEXO","S"
```

Una propiedad fundamental es que todas las instrucciones DATA de un programa forman un bloque único; equivalen, pues, a una única instrucción DATA con una lista de constantes que contenga todas las demás listas según el orden de aparición. Así, por ejemplo, si las tres DATA anteriores pertenecen a un mismo programa son equivalentes a:

```
6Ø    DATA    7,5,–2,1.9,18.43,5,6,"PUNTO",8,"LILA","FLEXO","S".
```

Cuando en un programa aparece la primera instrucción READ, lo cual ha de ocurrir antes de que se use alguna de las variables que define, el control va al bloque DATA y comienza a asignar, por orden, en primer valor a la primera variable, el segundo valor a la segunda variable, etc.

Como es lógico, debe haber una correspondencia de tipos, entre los valores leídos y las variables a las que son asignados.

Estos valores que han sido leídos del bloque DATA, podemos imaginar que son borrados de la lista DATA, pues al aparecer una nueva instrucción READ, se continúan asignando valores a partir del último asignado.

EJEMPLO 1

```
5Ø READ A,B,X$
7Ø READ U,V$T(1)
8Ø READ T(2),T(3),$(1)
...
...
15Ø DATA 7,8,"LUZ",24.7,"HOJA"
16Ø DATA 34,57,18,"LAPIZ"
```

en este caso, la asignación es la siguiente:

```
A = 7    B = 8    X$ = "LUZ"    U = 24.7    V$ = "HOJA"
T(1) = 34    T(2) = 57    T(3) = 18    A$(1) = "LAPIZ"
```

Evidentemente, si el primer READ fuese:

```
5Ø READ A,B,C
```

daría error, pues asignaría a C una cadena.

También daría error si la lista DATA es más corta que la lista de variables que solicitan valores, porque habría variables que exigen tomar un valor que no pueden conseguir. Por otra parte, si la lista es más amplia de lo necesario, es decir, hay más datos que variables, no da error, solo que hay valores que no son asignados a ninguna variable y su existencia es ignorada.

Las sentencias DATA pueden situarse en cualquier lugar del programa, aunque por comodidad suele escribirse todo el bloque al final, antes del END.

La instrucción DATA no tiene que estar necesariamente dispuesta para un READ concreto, aunque se permite, por supuesto. Veamos algunos ejemplos:

EJEMPLO 2

```
...
...
 4Ø IF A>Ø THEN 6Ø
 5Ø READ A,B$,C
 6Ø READ P,Q$,R
...
...
2ØØ DATA 3,"LAB",18.7,45
21Ø DATA "TOR",3.5,16,"CARO",4,87
...
...
```

La asignación será:

si A $\leqslant$ 0 A = 3 B$ = "LAB" C = 18.7
P = 45 Q$ = "TOR" R = 3.5

y si A > 0 P = 3 Q$ = "LAB" R = 18.7

los demás valores del bloque DATA son ignorados.

EJEMPLO 4

Buscar el menor de una lista de N números.

Este ejemplo ya ha sido hecho con anterioridad, usando para almacenar los N números un vector de N componentes. En esta ocasión estos N números figurarán en una DATA, siendo el esquema del programa esencialmente el mismo.

El valor de N es el primer valor leído por un READ y lo pondremos en una DATA aparte para evitar confusiones.

El siguiente programa busca el menor de un conjunto de 18 datos:

```
 1Ø READ N
 2Ø READ M
 3Ø FOR I=2 TO N
 4Ø READ A
 5Ø IF M>A THEN M=A
 6Ø NEXT I
 7Ø PRINT "EL MENOR DE LA LISTA ES ";M
 8Ø DATA 18
 9Ø DATA 2,3,4,6,5,4,7,8,3,4,5,6,7,8,2,3,4,3
1ØØ END
```

La instrucción 10 asigna a N el valor 18 que corresponde al número de datos de la lista; la instrucción 20 lee el primer dato de la lista y lo asigna a M; cada vez que lee un nuevo dato lo asigna a A; M representa siempre el menor de los datos leídos.

Observar que este programa es válido para cualquier lista de números sin más que cambiar el bloque DATA.

La ejecución de un programa es más rápida de esta forma que con INPUT, debido a que la instrucción INPUT requiere entrada manual de datos, lo cual es siempre más lento que la lectura directa.

16.2. LA INSTRUCCION RESTORE

Hemos visto cómo introducir datos a través de READ y DATA y cómo tenía que existir una correspondencia entre tipos de datos y variables. Los valores leídos de una DATA no vuelven a ser asignados a ninguna variable; cualquier asignación se hace siempre a partir del último valor leído.

La instrucción RESTORE, llamada de relectura, lo que hace es reactivar la DATA, es decir, la DATA vuelve a ser válida desde el principio y el siguiente dato a ser leído será, precisamente, el primero que aparezca en el bloque DATA.

EJEMPLO 1

```
20 READ A,B,C
30 RESTORE
40 READ X,Y,Z,T
...
...
100 DATA 7,9,5,3,4,6
```

la asignación es: A = 7 B = 9 C = 5 X = 7 Y = 9 Z = 5
T = 3
la instrucción 40 vuelve a leer en DATA desde el principio.

EJEMPLO 2

```
...
130 READ P,Q,R
140 IF B>1 THEN 160
150 RESTORE
160 READ X,Y,Z,T
...
...
200 DATA 3,6,7,9,10,12,1,3
```

Si B > 1 es P = 3 Q = 6 R = 7 X = 9 Y = 10 Z = 12
T = 1
y si B≤1 P = 3 Q = 6 R = 7 X = 3 Y = 6 Z = 7
T = 9

EJEMPLO 3

Los 12 datos que figuran en la DATA serán asignados a una matriz A(3,4) por filas y a una matriz B(4,3) por columnas:

```
10 DIM A(3,4,),B(4,3)
20 FOR I=1 TO 3
30 FOR J=1 TO 4
40 READ A(I,J)
50 NEXT J
60 NEXT I
70 RESTORE
80 FOR J=1 TO 4
90 FOR I=1 TO 3
100 READ B(I,J)
110 NEXT I
120 NEXT J
130 DATA 23,3,45,3,6,7,8,3,45,65,7,23
```

Las matrices A y B leídas son:

$$A = \begin{bmatrix} 23 & 3 & 45 & 3 \\ 6 & 7 & 8 & 3 \\ 45 & 65 & 7 & 23 \end{bmatrix}$$

$$B = \begin{bmatrix} 23 & 3 & 45 \\ 3 & 6 & 7 \\ 8 & 3 & 45 \\ 65 & 7 & 23 \end{bmatrix}$$

EJERCICIOS

1. Asignar los valores

2.4 45 35.6 "TELEX" "TOK" 6,7 67.8 4 "T" "----"

a las variables:

A(12) B C(1) B$ N$(4) P Q S$ M$(9)

2. Asignar 20 vaores leídos en una DATA a una matriz de cinco filas y cuatro columnas; releer estos valores para un vector de 20 componentes e imprimir ambos.

3. Escribir la impresión del siguiente programa según los valores de X:

```
10 INPUT X
IF X<1 THEN 50
30 READ A,B,C$
40 RESTORE
50 READ P,Q,R$
```

```
6Ø IF X=Ø THEN 3Ø
7Ø DATA 18,24,"PEPE",3,5,"LAR",14,13,"TIPO"
8ØPRINT A,B,C$,P,Q,R$,
9Ø END
```

3. Decir los valores que toman las variables A, B, C y D.

```
1Ø READ A,B,C
2Ø IF A<1 THEN 4Ø
3Ø READ A
4Ø READ B
5Ø DATA 2,3.5,18,45,37
```

17

Ficheros

17.1. GENERALIDADES SOBRE FICHEROS

En un centro de enseñanza se dispone de una ficha por cada alumno con sus datos personales y académicos las cuales están organizadas en un fichero; en general, un fichero es un conjunto organizado de datos entre los que existe algún tipo de relación.

Lo que identifica cada ficha es el nombre del alumno; sobre ella figuran, además, todos sus datos personales y académicos con la posibilidad de poder ir incorporando los nuevos resultados que se vayan produciendo. Sobre este fichero podemos hacer, por lo tanto, dos tipos de operaciones, de consulta y de actualización.

La organización del fichero puede variar según el uso que hagamos de él; en nuestro caso podemos hacerlo por orden alfabético general, o bien, por orden alfabético dentro de cada curso.

Los ficheros informáticos tienen una estructura semejante, con la particularidad de que el soporte de los datos es de carácter magnético, normalmente, cinta o disco.

La organización del fichero es el método de colocación de los datos sobre el soporte usado para el almacenamiento; dicha organización está íntimamente ligada al sistema de acceso y tratamiento de la información contenida en ese fichero; en informática suelen usarse tres tipos de ficheros atendiendo al modo de acceso:

1. Ficheros de acceso secuencial: son aquellos en los que para llegar a una determinada información, hay que pasar previamente por todo el fichero hasta llegar al punto buscado; el fichero anterior, formado por las fichas de alumnos ordenadas alfabéticamente, puede ser un ejemplo; otro ejemplo puede ser una cinta magnetofónica en la que para poder acceder a una posición hay que pasar antes por todas las que le preceden.
2. Ficheros de acceso directo: permiten acceder a cualquier información directamente,sin pasar por ninguna otra; esto exige que de cada información se tiene la posición donde se encuentra almacenada. Por ejemplo, sobre un disco musical, podemos acceder directamente a la parte que nos interese oír sin más que llevar la aguja al sitio adecuado; se supone, claro está, que conocemos la dirección donde se encuentra la melodía que nos interesa oír.

3. Los ficheros de acceso secuencial indexado: son un tipo intermedio entre los de acceso secuencial y los de acceso directo; disponen de un sistema de índices que apuntan a una zona donde se encuentra la información que nos interesa; una vez sobre esta zona, la búsqueda ha de ser secuencial. Tal puede ser el caso de buscar una palabra en una enciclopedia que dispone de un tomo para cada letra; la elección del tomo es un acceso directo, mientras que la búsqueda de la palabra de este tomo es un acceso secuencial.

17.2. TRATAMIENTO DE FICHEROS

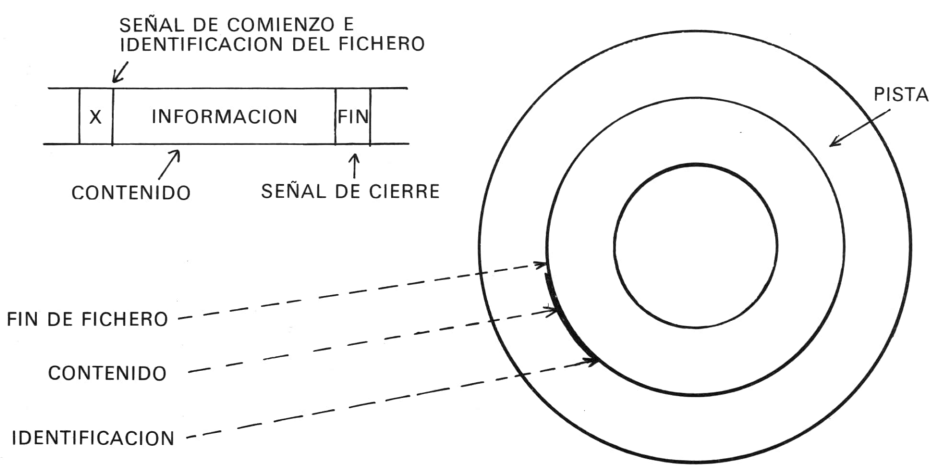
Cuando tenemos un cierto volumen de información y queremos guardarla en un fichero, debemos proceder a la creación de este fichero y a organizarlo para su posterior tratamiento; realmente, debe quedar muy claro, que la creación de un fichero tiene su única justificación en la medida en que después podamos acceder a la información que contiene, ya que de otra forma sería un trabajo inútil.

En el proceso de creación de un fichero tenemos al menos tres fases claramente diferenciadas:

1. *Definición:* todo fichero debe llevar una clave que permita reconocerlo con posterioridad; esta clave puede ser simplemente un nombre o incluir otras referencias respecto a su estructura interna y a su modo de acceso. Sobre la cinta o disco, esta clave se convierte en una señal indicativa e identificadora de que comienza dicho fichero.
2. *Suministro de información:* una vez definido el fichero, debemos proceder a suministrarle los datos que deba contener según una organizacion prefijada; esta información ocupará un espacio a partir de la señal de comienzo de fichero.

El contenido de un fichero no ha de ser necesariamente un conjunto de datos, sino que, como veremos más adelante, se puede almacenar programas en ellos.

3. *Cerrar el fichero:* incluye una señal indicativa de fin de fichero.



Sobre el fichero creado podemos hacer tres tipos de operaciones:

a) Verificación o comprobación de que ha sido creado correctamente.
b) Consultar posteriormente sobre alguna información.
c) Actualización de alguna de las informaciones contenidas en él.

17.3. FICHEROS DE ACCESO SECUENCIAL

A medida que aumenta la información contenida en un fichero se requieren soportes de almacenamiento más rápidos y de mayor capacidad: el almacenamiento secuencial en cinta magnética se desarrolló inicialmente para satisfacer esta necesidad.

Tal como hemos dicho en el apartado anterior, sobre la cinta habra una señal de identificación y comienzo de fichero, una zona de contenidos y otra señal de fin de fichero:

para acceder a uno de ellos hay que pasar por todos los anteriores.

El manejo de este tipo de ficheros es muy simple y permite almacenar grandes cantidades de información, lo que tiene como contrapartida la lentitud del acceso; la cinta magnética se limita, físicamente, a escribir y leer datos secuencialmente, de uno en uno, debido a la posición fija de la cabeza de lectura y escritura; además, sólo se puede acceder a información registrada en la dirección de avance.

Las instrucciones que permiten el uso de estos ficheros no son uniformes para todos los modelos de ordenadores, aunque son de uso frecuente las instrucciones OPEN, CLOSE, INPUT# y PRINT# que describimos seguidamente.

OPEN cualificadores: se usa para abrir y definir el fichero; los cualificadores deben ser, como mínimo, el nombre del fichero, una indicación de si es un fichero de lectura o escritura y un número que identifica el fichero para el caso de que un proceso deba mantener abiertos varios ficheros, es decir, podemos suponer que si un proceso necesita usar varios ficheros, transmite con cada uno de ellos a través de un canal de comunicación y que este número identifica el canal de transmisión.

Ejemplo: dos instrucciones OPEN pertenecientes a modelos distintos pero con efectos idénticos son las siguientes:

```
OPEN 5,1,"ALUMNOS"
OPEN "ALUMNOS" FOR INPUT AS FILE# 5
```

La primera de ellas asigna el canal 5 para un fichero de entrada (el 1 sería la clave de lectura) cuyo nombre es ALUMNOS; la segunda significa lo mismo pero no usa clave para indicar que el fichero sea de lectura.

CLOSE n: Cierra el fichero cuyo canal es *n* y que se supone fue abierto anteriormente.

INPUT#, n Lista de variables: lee del fichero de canal n datos que van siendo asignados, sucesivamente, a las variables que aparecen en la lista. Lógicamente, debe existir una correspondencia entre los tipos de los datos leídos y los de las variables a las que son asignados.

PRINT#, n Lista de variables: Escribe sobre el fichero los valores correspondientes a las variables que figuran en la lista.

Las anteriores instrucciones son usadas para la creación y uso de ficheros de datos; para los ficheros de programas se usan las instrucciones:

SAVE "nombre": con esta instrucción, los programas que existan en la memoria del ordenador son guardados en un fichero cuya identificación es el nombre que hayamos colocado entre las comillas.

LOAD "nombre": es lo inverso de SAVE; carga en la memoria del ordenador el fichero de programas definido con este nombre.

Para poder realizar un fichero secuencial usando las instrucciones anteriores, debemos cargarlo en la memoria del ordenador, hacerle allí las modificaciones deseadas y crear un nuevo fichero donde guardarlo, es decir, no podemos hacer modificaciones directamente sobre el fichero almacenado en cinta y cada actualización supone la creación de un nuevo fichero.

17.4. FICHEROS DE ACCESO DIRECTO

Sobre los ficheros de acceso secuencial hemos insistido en la lentitud con que podemos acceder a una determinada información; si se tiene que acceder a gran parte de un fichero, la cinta puede desempeñar un buen papel, pero si se trata de múltiples accesos, para hacer en cada caso alguna consulta o pequeñas actualizaciones, es mucho mejor un fichero de acceso directo sobre disco que permite localizar rápidamente los datos que nos interesen sin tener que leer y escribir el fichero de datos completo.

Imaginemos que una entidad bancaria tiene 50 000 cuentas corrientes y necesita con frecuencia ver el estado de alguna de ellas; evidentemente, el acceso secuencial es totalmente inútil en este caso. habría que crear un fichero sobre un sistema de discos donde se almacenarían las cuentas corrientes y se procedería a asignar a cada cuenta corriente un identificador de la dirección donde se encuentra; por ejemplo, podría ser 2 72 3, cuyo significado hipotético sería que la cuenta buscada se encuentra en el segundo disco, sobre la pista 72 en el sector 3. Ni que decir tiene que la realidad es algo más compleja.

El tratamiento y organización de ficheros de acceso directo es un tema mucho más complicado, sobre todo, en cuanto el volumen de información adquiere ciertas dimensiones y ha de ser consultada y actualizada con frecuencia; esto nos llevaría al estudio de las bases de datos que está fuera de las posibilidades de este libro.